

(21)(A1) **2,219,037**
(22) 1997/10/23
(43) 1998/04/25

(72) ASHBY, Richard A., US
(72) BOUZIDE, Paul M., US
(72) ISRANI, Vijaya S., US
(72) LAMPERT, David S., US
(72) NATESAN, Senthil K., US
(72) KILLEY, Grant S., US
(72) JASPER, John C., US
(72) FERNEKES, Robert P., US
(72) FEIGEN, Jerry S., US

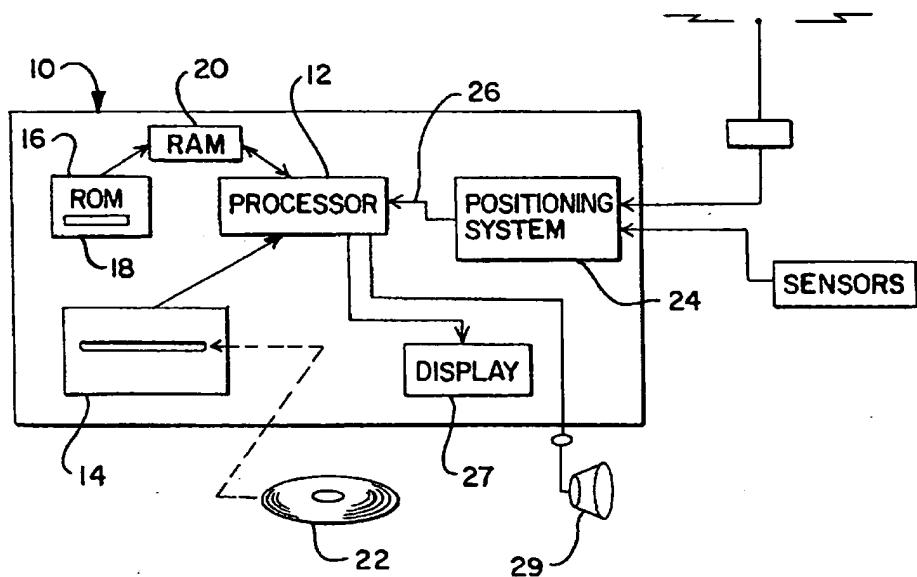
(71) Navigation Technologies Corporation, US

(51) Int.Cl. ⁶ G08G 1/0968, G08G 1/0969

(30) 1996/10/25 (08/740,298) US

(54) **COUCHE D'INTERFACE POUR SYSTEME DE NAVIGATION**

(54) **INTERFACE LAYER FOR NAVIGATION SYSTEM**



(57) L'invention porte sur une méthode et un système améliorés qui prévoient une couche d'interface d'accès aux données dans un système de navigation. Ce système de navigation comprend un programme de logiciel de navigation, offrant à un utilisateur du système des caractéristiques de navigation, et une base de données géographiques stockées sur un support de données lisible

(57) An improved method and system that provides for a data access interface layer in a navigation system. The navigation system is of the type that includes a navigation application software program that provides navigating features to a user of the system and a geographic database stored on a computer-readable storage medium wherein the geographical database

(21) (A1) **2,219,037**
(22) 1997/10/23
(43) 1998/04/25

par un ordinateur. La base de données géographiques comporte des renseignements sur la région géographique sur laquelle portent les caractéristiques de navigation offertes à l'utilisateur par le système de navigation. La couche d'interface d'accès aux données est entreposée de préférence dans le système de navigation en tant que bibliothèque des fonctions du logiciel, fonctionne conjointement avec le logiciel de navigation du système, isole le logiciel des données géographiques qui sont stockées sur le support de données, intercepte les interrogations sur les données géographiques exécutées par le logiciel, extrait les données géographiques du support de données et convertit les données dans un format que le logiciel peut lire, et offre une gestion de mémoire qui facilite l'accès et l'utilisation rapide et efficace des données géographiques stockées sur le support en question. En reconnaissant que différents types de supports présentent différents formats physiques, la couche d'interface d'accès aux données respecte et isole les différences, de sorte que les parties de la couche d'interface qui interagissent avec le logiciel de navigation puissent être génériques.

includes information relating to the geographical region about which the navigation system provides the navigation features to the user. The data access interface layer is preferably stored in the navigation system as a library of software functions. The data access interface layer operates in conjunction with the navigation system application software. The data access interface layer isolates the navigation application software from the geographic data which are stored on the storage medium. The data access interface layer intercepts requests by the navigation application software for geographic data. The data access interface layer retrieves geographic data from the storage medium and converts the data into a format usable by the navigation application software. The data access interface layer also provides for memory management that facilitates accessing and using geographic data from the particular storage medium quickly and efficiently. By recognizing that different media types have different physical formats, the data access interface layer accommodates and isolates the differences so that the portions of the data access interface layer that interact with the navigation application software can be generic.



ABSTRACT

2 An improved method and system that provides for a data access
3 interface layer in a navigation system. The navigation system is of the type
4 that includes a navigation application software program that provides navigating
5 features to a user of the system and a geographic database stored on a
6 computer-readable storage medium wherein the geographical database includes
7 information relating to the geographical region about which the navigation
8 system provides the navigation features to the user. The data access interface
9 layer is preferably stored in the navigation system as a library of software
10 functions. The data access interface layer operates in conjunction with the
11 navigation system application software. The data access interface layer isolates
12 the navigation application software from the geographic data which are stored
13 on the storage medium. The data access interface layer intercepts requests by
14 the navigation application software for geographic data. The data access
15 interface layer retrieves geographic data from the storage medium and converts
16 the data into a format usable by the navigation application software. The data
17 access interface layer also provides for memory management that facilitates
18 accessing and using geographic data from the particular storage medium quickly
19 and efficiently. By recognizing that different media types have different
20 physical formats, the data access interface layer accommodates and isolates the
21 differences so that the portions of the data access interface layer that interact
22 with the navigation application software can be generic.

1
2 **INTERFACE LAYER**
3 **FOR NAVIGATION SYSTEM**
4
5
6
7
89 **BACKGROUND OF THE INVENTION**
10

11 The present application relates to navigation systems and in particular,
12 the present application relates to a navigation system interface layer that
13 facilitates use and access of a geographic database stored on a physical storage
14 medium.

15 Computer-based navigation systems for use on land have become
16 available in a variety of forms and provide for a variety of useful features. One
17 exemplary type of navigation system uses (1) a detailed data set of one or more
18 geographic areas or regions, (2) a navigation application program, (3)
19 appropriate computer hardware, such as a microprocessor, memory, and storage,
20 and, optionally, (4) a positioning system. The detailed geographic data set
21 portion of the navigation system is in the form of one or more detailed,
22 organized data files or databases. The detailed geographic data set may include
23 information about the positions of roads and intersections in or related to one or
24 more specific geographic regional areas, and may also include information
25 about one-way streets, turn restrictions, street addresses, alternative routes,
26 hotels, restaurants, museums, stadiums, offices, automobile dealerships, auto
27 repair shops, etc.

28 The positioning system may employ any of several well-known
29 technologies to determine or approximate one's physical location in a
30 geographic regional area. For example, the positioning system may employ a
 GPS-type system (global positioning system), a "dead reckoning"-type system,

1 or combinations of these, or other systems, all of which are well-known in the
2 art.

3 The navigation application program portion of the navigation system is a
4 software program that uses the detailed geographic data set and the positioning
5 system (when employed). The navigation application program may provide the
6 user with a graphical display (e.g. a "map") of the user's specific location in the
7 geographic area. In addition, the navigation application program may also
8 provide the user with specific directions to locations in the geographic area
9 from wherever the user is located.

10 Some navigation systems combine the navigation application program,
11 geographic data set, and optionally, the positioning system into a single unit.
12 Such single unit systems can be installed in vehicles or carried by persons.
13 Alternatively, navigation application programs and geographic datasets may be
14 provided as software products that are sold or licensed to users to load in their
15 own personal computers. In further alternatives, the navigation system may be
16 centrally or regionally located and accessible to multiple users on an "as
17 needed" basis, or alternatively, on-line via a network or communications link.
18 Personal computer-based systems may be stand-alone systems or may utilize a
19 communication link to a central or regional or distributed system. Also, users
20 may access a navigation system over an on-line service such as the Internet, or
21 over private dial-up services, such as CompuServe, Prodigy, and America
22 Online. In-vehicle navigation systems may use wireless communication
23 connections. Navigation systems may also be used by operators of vehicle
24 fleets such as trucking companies, package delivery services, and so on.
25 Navigation systems may also be used by entities concerned with traffic control
26 or traffic monitoring.

27 Computer-based navigation systems hold the promise of providing high
28 levels of navigation assistance to users. Navigation systems can provide
29 detailed instructions for travelling to desired destinations, thereby reducing
30 travel times and expenses. Navigation systems also can provide enhanced
31 navigation features such as helping commuters and travellers avoid construction

1 delays and finding the quickest routes to desired destinations. Navigation
2 systems can also be used to incorporate real-time traffic information.

3 In order to provide these useful and enhanced features in a navigation
4 system, there is a need to gather and organize comprehensive, detailed, reliable,
5 and up-to-date data about geographical regions and areas. There is also a need
6 to continuously update the geographic data since many data can rapidly become
7 out-of-date. Presently, the collection of such geographic data and the provision
8 of such data in a computer-usable format are provided by Navigation
9 Technologies of Sunnyvale, California.

10 One potential problem associated with providing enhanced features and
11 updated geographic databases for navigation systems is that there are numerous
12 different navigation system platforms. These platforms may have different
13 hardware, navigation software, operating systems, and so on. Many of these
14 different platforms have developed independently and are incompatible with
15 each other. Thus, even if it were possible to gather updated geographic data
16 and produce an updated computer-readable geographic database, it becomes
17 difficult to provide and distribute geographic databases that can be used with
18 the various different types of navigation system platforms due to the numerous
19 different platforms that exist. This problem may be exacerbated in the future as
20 navigation system manufacturers develop new generations of navigation systems
21 with more and enhanced features.

22 Another problem exists with regard to providing updated geographic
23 data for existing navigation systems. Just like conventional printed maps,
24 geographic information used in computer-based navigation systems can become
25 out-of-date. For example, new roads are built, businesses change locations,
26 road construction closes roads, detours are established, museum and restaurant
27 hours change, etc. It is expected that end-users, such as vehicle owners who
28 have navigation systems in their vehicles, will want to have the geographic data
29 in their navigation systems updated from time to time.

30 The proliferation of multiple, different, incompatible navigation system
31 platforms, mentioned above, also presents an obstacle to providing updated
32 geographic data for end-users, i.e. the persons and businesses who own and use

1 navigation systems. In order to provide updated geographic data for an end-
2 user's navigation system over the lifetime of the navigation system, the provider
3 of geographic data needs to provide a product that not only has updated
4 geographic data, but also that is compatible with the user's particular navigation
5 system. Over the expected lifetime of navigation systems, which may be ten
6 years or more, this would require supporting a growing number of old,
7 incompatible navigation systems and platforms. If specialized versions of
8 updated geographic data products had to be produced for each separate,
9 different type or version of navigation platform, the number of different
10 products would continue to increase over time thereby making the provision of
11 updated geographic data products to end-users difficult and expensive.

12 Accordingly, it is an object of the present invention to provide a solution
13 to the problem of providing geographic data for a variety of hardware
14 platforms.

15 Further, it is an object of the present invention to provide an improved
16 navigation system and geographic navigation database(s) for use therein, that
17 can be efficiently developed, manufactured, customized, distributed, and/or
18 updated across a variety of navigation system platforms, operating systems, and
19 releases.

20

21 SUMMARY OF THE INVENTION

22 To achieve the foregoing and other objectives and in accordance with
23 the purposes of the present invention, there is provided an improved method
24 and system for a data access interface layer in a navigation system. The
25 navigation system is of the type that includes a navigation application software
26 program that provides navigating features to a user of the system and a
27 geographic database stored on a computer-readable storage medium wherein the
28 geographical database includes information relating to the geographical region
29 about which the navigation system provides the navigating features to the user.
30 The data access interface layer is preferably stored in the navigation system as a
31 library of software functions. The data access interface layer operates in
32 conjunction with the navigation system application software. The data access

1 interface layer isolates the navigation application software from the geographic
2 data which are stored on the storage medium. The data access interface layer
3 intercepts requests by the navigation application software for geographic data.
4 The data access interface layer retrieves geographic data from the storage
5 medium and converts the data into a format usable by the navigation application
6 software. The data access interface layer also provides for memory
7 management that facilitates accessing and using geographic data from the
8 particular storage medium quickly and efficiently. By recognizing that different
9 media types have different physical formats, the data access interface layer
10 accommodates and isolates the differences so that the portions of the data
11 access interface layer that interact with the navigation application software can
12 be generic.

13 **BRIEF DESCRIPTION OF THE DRAWINGS**

14 FIG. 1 is a diagram illustrating a navigation system including a storage
15 medium upon which geographical data, and optionally other data, are stored.

16 FIG. 2 is diagram illustrating the software components in the navigation
17 system of FIG. 1.

18 FIG. 3 is a block diagram illustrating the software components of the
19 navigation system of FIG. 1 including the major systems of the interface layer
20 of FIG. 2.

21 FIG. 4 is a block diagram illustrating one embodiment of the resource
22 manager software architecture of FIG. 3.

23 FIGS. 5A and 5B are illustrations representing memory storage during
24 operation of the cursor management system shown in FIG. 3.

25 FIG. 6 is an example illustrating a parcel ID used in the physical
26 address storage mapper of FIG. 4.

27 FIG. 7 is a flow diagram illustrating the process for linking the
28 navigation application component with the data interface layer.

29 FIG. 8 is a block diagram illustrating another embodiment of the
30 resource manager software architecture of FIG. 3.

1 DETAILED DESCRIPTION OF THE PRESENTLY
2 PREFERRED EMBODIMENTS

3 I. Overview of navigation system

4 Referring to FIG. 1, there is a diagram illustrating an exemplary
5 configuration of a navigation system 10. The navigation system 10 is a
6 combination of hardware and software components. In one embodiment, the
7 navigation system 10 includes a processor 12, a drive 14 connected to the
8 processor 12, and a memory storage device 16, such as a ROM, for storing a
9 navigation application software program 18. The navigation application
10 software program 18 is loaded from the ROM 16 into a memory 20 associated
11 with the processor 12 in order to operate the navigation system. The processor
12 12 may be of any type used in navigation systems, such as 32-bit processors
13 using a flat address space, such as a Hitachi SH1, an Intel 80386, an Intel 960,
14 a Motorola 68020 (or other processors having similar or greater addressing
15 space). Processor types other than these, as well as processors that may be
16 developed in the future, are also suitable. A storage medium 22 is installed in
17 the drive 14. In one present embodiment, the storage medium 22 is a
18 CD-ROM. In another alternative embodiment, the storage medium 22 may be a
19 PC Card (PCMCIA card) in which case the drive 14 would be substituted with
20 a PCMCIA slot. Various other storage media may be used, including fixed or
21 hard disks, DVD (digital video disks) or other currently available storage
22 media, as well as storage media that may be developed in the future. The
23 storage medium 22 includes geographic data, as described more fully in the
24 copending application entitled "IMPROVED SYSTEM AND METHOD FOR
25 USE AND STORAGE OF GEOGRAPHIC DATA IN PHYSICAL MEDIA",
26 referenced above.

27 The navigation system 10 may also include a positioning system 24.
28 The positioning system 24 may utilize GPS-type technology, a dead reckoning-
29 type system, or combinations of these, or other systems, all of which are known
30 in the art. The positioning system 24 outputs a signal 26 to the processor 12.
31 The signal 26 may be used by the navigation application software 18 that is run
32 on the processor 12 to determine the location, direction, speed, etc., of the
33 navigation system 10. The navigation system 10 uses the geographic data

1 stored on the storage medium 22, possibly in conjunction with the output 26
2 from the positioning system 24, to provide various navigation application
3 features. These navigation application features may include route calculation,
4 map display, vehicle positioning (e.g. map matching), maneuver generation
5 (wherein detailed directions are provided for reaching a desired destination),
6 and other features. These navigation application features are provided by
7 navigation application programs (i.e. subprograms or functions) that are part of
8 the navigation application software program 18. The navigation features are
9 provided to the user (e.g., the vehicle driver) by means of a display 27,
10 speakers 29, or other means.

11 Referring to FIG. 2, the navigation application software program 18
12 includes separate applications (or subprograms) 200. The navigation application
13 programs 200 of the navigation system 10 may be regarded as including route
14 calculation functions, maneuver generation functions, map display functions,
15 vehicle positioning functions, destination resolution capabilities, and so on.
16 FIG. 2 shows these separate navigation applications 200 including a route
17 calculation function 28, a map display function 30, a maneuver generation
18 function 32, and other functions or subprograms 34. Although these navigation
19 application subprograms are represented as separate functions or applications
20 within the navigation application software program 18, these functions may be
21 combined and provided as a single program.

22 In FIG. 2, the storage medium 22 is shown to have geographic data 40
23 stored on it. The geographic data 40 are in the form of one or more computer-
24 readable data files or databases. The geographic data 40 may include
25 information about the positions of roads and intersections in or related to a
26 specific geographic regional area, and may also include information about one-
27 way streets, turn restrictions, street addresses, alternative routes, hotels,
28 restaurants, museums, stadiums, offices, automobile dealerships, auto repair
29 shops, etc. The geographic data 40 are organized in a format and stored on the
30 storage medium 22 in a manner that facilitates use and access by the navigation
31 application functions 200 of the navigation application program 18. The
32 organization and storage of the geographic data 40 are described in more detail

1 in the above referenced copending application entitled "IMPROVED SYSTEM
2 AND METHOD FOR USE AND STORAGE OF GEOGRAPHIC DATA ON
3 PHYSICAL MEDIA."

4 The various separate navigation applications 200 of the navigation
5 application software 18 access and read portions of the geographic data 40 from
6 the storage medium 22 in order to provide useful navigation features to the user
7 of the navigation system 10. In a present embodiment, a data access interface
8 layer 41 is located between the various navigation applications 200 (such as the
9 functions 28, 30, 32, and 34) and the geographic dataset 40. The data access
10 interface layer 41 isolates the navigation application functions 200 from the
11 formatting, storage, and other details of the geographic data 40. Accordingly,
12 the data access interface layer 41 provides for various different navigation
13 system platforms to use a common geographic database product, i.e. the
14 geographic dataset 40 stored on the storage medium 22. In addition, the data
15 access interface layer 41 facilitates allowing an end-user to update the
16 geographic data in the end-user's navigation system.

17 **II. Data access interface layer - overview**

18 The data access interface layer 41 is a software component that may
19 reside in the navigation system 10. In a preferred embodiment, at least a
20 portion of the data access interface layer 41 is linked or compiled into the
21 executable module that forms the navigation application software 18 in the
22 navigation system 10 that is loaded into the memory 20 (FIG. 1) from the ROM
23 16 when the navigation system 10 is operated. Within the data access interface
24 layer 41 there are several subsystem components. There are internal interfaces
25 between these subsystem components and external interfaces to the navigation
26 application software components 200 and operating system 202 of the
27 navigation system 10. Data are presented to the navigation applications as
28 logical data entities. The logical data model entity records are fixed length and
29 contain decompressed data without any of cross-reference information.

30 In a preferred embodiment, the data access interface layer 41 is provided
31 in the form of a library of software functions. This library of functions

1 provides data access for use by the various components or subprograms 200 of
2 the navigation application software program 18. In one preferred embodiment,
3 some or all of these library functions are directly linked to the various
4 navigation applications 200 to form the navigation software product 18. Thus,
5 the data access interface layer 41 is incorporated into the navigation application
6 software of the navigation systems of OEM's (original equipment
7 manufacturers) or aftermarket automotive navigation systems, which use a
8 separately stored geographic database. In alternative embodiments, discussed
9 below, the data access interface layer 41 may be incorporated into other-than-
10 in-vehicle navigation systems.

11 In a preferred embodiment, the source code is written in the C
12 programming language, although in alternate embodiments, other languages
13 may be used.

14 Because the data access interface layer 41 is used with various different
15 navigation systems, the interface layer 41 takes into account differences among
16 these systems. Some in-vehicle navigation systems have relatively small
17 quantities of RAM, slow I/O devices, and proprietary and/or real-time-oriented
18 operating system kernels. Some of these navigation systems calculate an
19 optimal route and provide real-time, turn-by-turn guidance to the end-user.
20 Accordingly, it is advantageous to integrate various position and heading sensor
21 information in real-time and in the background. Some of these navigation
22 systems also provide cartographic display, a flexible capability to obtain route
23 destination points, and an interface oriented to in-vehicle ergonomics.

24 As mentioned above, the data access interface layer 41 isolates the
25 navigation application software 18 from the size, complexity, and evolution of
26 the geographic map database 40 on the physical medium 22. In a preferred
27 embodiment, similar or identical data access interface layers can be used by
28 different navigation systems. The data access interface layer 41 is portable
29 across different hardware platforms. The data access interface layer 41
30 provides versatility to support most or all envisioned navigation application
31 functionality for a wide range of product capabilities and hardware platforms.
32 In a preferred embodiment, the software library that comprises the data access

1 interface layer uses less than 256 K bytes of memory and 16 Kbytes of stack
2 memory, and at least 256 Kbytes of heap memory.

3 As mentioned above, the geographic data 40 is stored on the storage
4 device 22, such as a CD-ROM. In a preferred embodiment, the geographic data
5 is stored using some or all of the physical storage format features disclosed in
6 the above referenced copending application entitled "IMPROVED SYSTEM
7 AND METHOD FOR USE AND STORAGE OF GEOGRAPHIC DATA IN
8 PHYSICAL MEDIA." The features used in the physical storage format provide
9 for efficient access to the geographic data and associated third party data
10 ("TPD"). Different types of storage media have distinct data capacities and
11 access characteristics. Accordingly the physical storage format features
12 disclosed in the copending application account for the various media intended
13 for use in navigation systems. Although the present embodiment of the data
14 access interface layer 41 uses the physical storage format disclosed in the
15 copending application, some or all of the features of the data access interface
16 layer disclosed herein may be used with other formats.

17 As described in the copending application, the geographic data are stored
18 on the medium by means of a geographic dataset compiler. The compiler
19 accepts geographic data and associated third party data in an interchange
20 format. The geographic data and third party data are input to the compiler
21 which generates an output in the form of an appropriate physical storage
22 format. The geographic data may be published by Navigation Technologies of
23 Sunnyvale, California.

24 FIG. 3 is a block diagram of the navigation system 10 showing
25 components of the data access interface layer 41. In FIG. 3, the data access
26 interface layer 41 is shown in the navigation application software 18 logically
27 below the navigation application programs 200 and above the operating system
28 202. This allows the data access interface layer 41 to be responsive to data
29 access requests from the various navigation application software programs 200.

30 The software library that forms the data access interface layer 41 may be
31 regarded as being composed of three major subsystems. The topmost
32 subsystem is the data access programming interface query logic ("DQL" or

1 "query logic") subsystem 210. The query logic subsystem 210 provides a
2 function call interface 212 to the navigation application software programs 200.
3 The data access interface layer 41 defines a data structure view (referred to as
4 the "Logical Data Model" or "LDM") of the geographic data 40 on the storage
5 medium 22 to the navigation application programs 200.

6 In a presently preferred embodiment, the data structure view defined by
7 layer 41 is in the C-language. As mentioned above, the logical data model
8 represents the entities in fully decompressed form. The entities in the logical
9 data model form are not compacted. Although this results in wasted space
10 within data entities, it facilitates immediate use of the data entities by the
11 navigation application. Each of the data entities of a given type may have a
12 fixed, known predetermined size. Also, in a preferred embodiment, entities in a
13 logical data format include no cross-reference information or other types
14 information that require processing before the data can be used by the
15 navigation application.

16 The query logic subsystem 210 includes a set of function calls that allow
17 the navigation application programs 200 to request a particular set of entities in
18 the logical data model format. The query logic subsystem 210 also implements
19 the logic that locates the requested entities and manages the resulting data set
20 which is then returned to the requesting component of the navigation
21 application programs 200.

22 The query logic subsystem 210 resolves data access queries in terms of
23 subdivisions of the physical media, called "parcels", which contain the requested
24 data in the physical storage format of the geographic data 40 on the medium
25 22. The query logic subsystem 210 also provides for management of the query
26 result set, which can be expressed in terms of a "cursor" for those queries that
27 can return more than one record. The cursor is a construct that allows the
28 navigation application programs to access parts of a large query result set
29 without requiring an in-memory copy of each record in translated logical data
30 model form.

31 Below the query logic subsystem 210 is an index management and data
32 translation (IMN) subsystem 216. The index management and data translation

1 subsystem 216 provides several separate functions. The first function uses
2 index information to locate data for physical entities stored on the medium as
3 directed by the query logic subsystem 210. A second function provided by the
4 index management and data translation subsystem 216 is to unpack or otherwise
5 decompress the optimized entities and to transform the entities into the logical
6 data model data entities that are returned to the requesting navigation
7 application program 200. Where the geographic data 40 stored on the medium
8 are packed or compressed, the second function provided by subsystem 216 also
9 serves to unpack or otherwise decompress the optimized physical entities so that
10 they may be transformed to the logical data model format. Another function
11 provided by the index management and data translation subsystem is the
12 provision for forward/backward compatibility across different versions, as
13 explained further below.

14 Below the index management and data translation subsystem 216 is the
15 resource management subsystem 220. As mentioned above, in some navigation
16 systems, the amount of memory is relatively small and the I/O bandwidth is
17 relatively low. Conventional techniques for the management of these resources
18 may be limited. For example, one approach used in some navigation systems to
19 solve performance problems due to slow I/O is to use memory to cache data,
20 thereby minimizing physical I/O. But memory in these types of navigation
21 systems may be limited as well, particularly where multiple functions of the
22 navigation application 200 may need to operate concomitantly.

23 An improved approach to resource management is provided by the
24 present embodiment. In the present embodiment, the data access interface layer
25 41 is provided with its own resource management subsystem 220 that is
26 separate from any resource management function provided by the navigation
27 application programs 200. The interface layer 41 is provided with a portion of
28 the navigation system memory for its exclusive use and which is managed by
29 the resource management subsystem 220. In addition, the resource management
30 subsystem 220 has the capability to use additional portions of navigation system
31 memory when they are not needed by the navigation application programs. The
32 resource management subsystem 220 provides for improved management of

1 memory and I/O by focussing on data access requirements of the query logic
2 subsystem taking into account the specific data organization of the physical
3 storage format. The resource management subsystem 220 mediates access to
4 cache memory buffers and the physical I/O channel for data access tasks.
5 Management of these resources also benefits from coordination of actions and
6 data needs above the level of the data access interface layer 41, i.e. in the
7 software of the navigation application programs 200. The data access needs of
8 the navigation application programs 200 are used to help determine which data
9 to retain in cache memory. An advantageous way to provide this information
10 to the resource management subsystem 220 is through additional access
11 program parameters that allow the data access interface layer 41 to prepare data
12 in the background for anticipated future use. This is in addition to data access
13 calls where the navigation application program waits for immediately required
14 data. In addition, the resource management subsystem 220 may accept
15 parameters that allow for controlling the order of data access from the physical
16 medium.

17 Each of these interface layer subsystems is discussed in more detail
18 below.

19 **III. Query Logic (DQL) Subsystem 210**

20 The query logic subsystem 210 represents one level of the software
21 library that forms the data access interface layer 41. The query logic subsystem
22 210 implements an interface 212 to the navigation application programs 200 by
23 providing a view of the geographic database to the navigation application 200.
24 In a present embodiment, the data access interface 41 provides for data access.
25 The data access interface 41 defines a set of C-language data structures (the
26 logical data model) and a set of function calls that deliver these logical data
27 model entities to the navigation application software programs 200, which use
28 the function calls to access the geographic data. (Although a preferred
29 embodiment of the data interface layer 41 is defined as C-language structures,
30 the interface layer could be defined in any suitable programming language). In
31 a preferred embodiment, each of the data access interface layer function calls

1 made by the navigation application programs is implemented as a wrapper
2 around a corresponding function call in the query logic subsystem 210.

3 In general, the query logic subsystem 210 provides three kinds of data
4 access capability to the navigation application programs. One kind of data
5 access provided by the query logic subsystem allows the navigation application
6 programs to request a particular data entity by its database identifier (DBID).
7 The query logic system 210 provides the requested data entity to the navigation
8 application in the logical data model format. Another kind of data access
9 provided by the query logic subsystem 210 allows the navigation applications to
10 request a set of data entities that are related to a particular data entity. The
11 query logic subsystem provides the navigation application with a cursor
12 (explained in more detail below) to the resultant set of data entities. Some or
13 all of the data entities in the resultant set may be in the logical data format. A
14 third kind of data access provided by the query logic subsystem allows the
15 navigation application to obtain data based upon a search request. The query
16 logic subsystem returns a cursor to a resultant set of data entities that match the
17 search criteria.

18 The functions that form the interface 212 allow the navigation
19 application programs 200 to request a particular entity or a group of entities
20 (such as "nodes", "places", "segments", "points of interest", "postal codes", and
21 so on) which can be qualified by a particular subset of attributes (such as "all
22 segments that have an end point at *node X*", or "all municipalities including the
23 word "*Lake*" in name, and so on) which are either directly a part of or are
24 otherwise closely associated with the entity. Alternatively, these requests can
25 also be qualified by geographical parameters or attributes, such as all segments
26 within a specified rectangular area or inside a particular named place. The
27 geographic qualifiers can be applied to the primary map data entities and can be
28 useful for narrowing down a search for desired data.

29 The functions in the query logic subsystem 210 provide for pervasive
30 access to data. This means that access to any and all logical data model entities
31 is supported with a reasonable level of performance regardless of navigation
32 application intent. For example, the route guidance software 28 (FIG. 2) may

1 require access to point-of-interest data. Support for pervasive access to data
2 occurs regardless of the classification of entities that occurs at the physical
3 storage format level to optimize data access performance to the base set of
4 entities required for important functions such as route calculation or map
5 display. Pervasive access to data offers some synergy with the geographic
6 query qualifiers. For example, rectangular queries are commonplace for
7 geometric data such as segments or nodes, but are also useful for street names
8 and points-of-interest.

9 The query logic subsystem 210 also includes the ability to initiate a data
10 access transaction that is predictive in nature, e.g. where data are not required
11 right away but are anticipated to be needed soon. This type of data access
12 transaction preferably occurs in the background, allowing the navigation
13 application programs to continue to perform work. As such, the query logic
14 subsystem 210 provides the capability to make these predictive access calls in
15 addition to normal access calls where the data are required immediately. This
16 function coordinates with the navigation application programs 200 to predict
17 what data may be required next.

18 Some of the query logic subsystem functions return a single entity or
19 additional detail about a particular entity. However, others of the functions
20 return an unpredictable number of entities. Depending on the particular query
21 and the degree to which it is qualified, this number could potentially be quite
22 large. For this reason, these query logic subsystem functions include a cursor
23 management subsystem 249. A cursor is defined at the time of the query. The
24 cursor forms a window (explained below) into an arbitrarily large result set
25 built by the query. The cursor allows the navigation application to specify how
26 many entities should be returned at a time when data are fetched through the
27 cursor. General-purpose cursor management functions in the data access
28 programming interface 212 allow the navigation application software programs
29 200 to subsequently move this window in a flexible and convenient manner.

1 A. **Query resolution**

2 The data query logic subsystem 210 implements the interface to the
3 navigation application programs 200 by resolving queries from the programs to
4 a set of physical data subdivisions, called parcels, which exist on the media.
5 These parcels may contain spatially-organized data, non-spatial data, index
6 information, or other data. Once parcels are identified in response to a query,
7 they are read into memory. The data within a parcel may be further organized
8 into subsets to minimize the amount of data that need to be examined in a
9 parcel to resolve a query. The appropriate parcels or parcel data subsets are
10 identified and entities are located within those parcels or parcel data subsets in
11 order for the data query logic subsystem 210 to build a cursor result set for a
12 particular request. Alternatively, a single logical data record may be returned
13 directly to the navigation application software programs 200 for simpler queries
14 which always return a single record and which do not require construction of a
15 cursor. However, both of these approaches locate and obtain one or more
16 physical parcels, which ultimately reside in the physical storage format on the
17 storage medium 22, and physically examine some subset of the parcel data to
18 locate the entities that make up the result set. Once the result set is identified,
19 the entities are transformed into the logical data model format and returned to
20 the navigation application software programs 200.

21 Resolving queries requires information about the particular physical
22 storage format used on the storage medium 22. The data access interface layer
23 41 isolates the parts of its software library that are dependent upon the specific
24 physical storage format allowing other components of its library to be
25 independent of a particular physical format. The components that are
26 dependent upon the storage format are included in the index management and
27 data translation subsystem 216. This subsystem 216 includes two further
28 subsystems: the index management and navigation subsystem 242 and the
29 physical-to-logical data translation subsystem 244.

30 At least two approaches are used to minimize accesses to the physical
31 medium. One approach is to use index information within parcels to look up
32 data. Another approach is to use data entity ID's that explicitly identify the

1 parcel in which the data entity is located. (This latter approach is described in
2 more detail in the above-referenced copending application.) Both index
3 information and explicit data entity ID's are used to resolve a query to the
4 parcel, subdivision of a parcel, or record level. The location and type of index
5 information that is available depend on the physical storage format. It follows
6 also that the algorithm for navigating the index also depends on the physical
7 storage format. The function for navigating the index on the physical medium
8 is included in the index management and navigation subsystem 242. Similarly,
9 the transformation of compressed data from the format on the physical media
10 into a form amenable for query resolution and translation to the logical data
11 model format for return to the navigation application program is included in the
12 physical-to-logical data transformation subsystem 244.

13 The query resolution process provided by the query logic subsystem 210.
14 depends for data access upon the format-dependent services provided by the
15 index management and navigation subsystem 242 and the physical-to-logical
16 subsystem 244. This allows the implementation of the query logic subsystem
17 to be generic and independent of the physical storage format. The query
18 resolution process also makes use of lower-level services to obtain parcels from
19 the physical medium or cache memory and to obtain private dynamic memory
20 buffers. These services are provided by the resource management subsystem
21 (SRM) 220. The index management and navigation subsystem 242, the
22 physical-to-logical subsystem 244, and the resource management subsystem 220
23 build a foundation for the query resolution process and appear logically below
24 the level of the data query logic subsystem 210 in FIG. 3. This layered
25 approach defines additional interfaces between the data query logic subsystem
26 210 and both the index management and navigation subsystem 242 and the
27 physical-to-logical subsystem 244.

28 One interface 248 in the index management and navigation subsystem
29 242 takes an index specifier and query parameters information from the data
30 query logic subsystem 210 and returns a set of parcel identifiers. A parcel
31 identifier is a format-specific construct for identifying the parcel that is related
32 to the physical organization of the data in the physical storage format on the

1 medium. The parcel identifiers are then passed to the resource management
2 subsystem 220 to obtain a pointer to the physical parcel in memory. Another
3 interface 250 in the index management and navigation subsystem 242 provides
4 for the translation from a specific entity identifier passed by the navigation
5 application program 200 via the data query logic subsystem 210 to an identifier
6 for the parcel which contains that entity. To further resolve a query to a
7 particular parcel data subset or record, additional interfaces 253 between the
8 data query logic 210 and index management and navigation subsystem 242 are
9 required. The query parameters passed via this interface 253 are used by the
10 index management and navigation subsystem 242 to navigate the internal index
11 of the parcel (i.e. the index for the parcel data subsets) to locate the
12 corresponding parcel data subset or record.

13 When internal index information is not present to resolve a query to a
14 particular entity or set of entities within the parcel, a brute-force record
15 inspection or binary search techniques are provided at the level of the data
16 query logic subsystem 210. When these approaches are used, the entities are
17 represented in an decompressed intermediate format ("DIF") that allows
18 attribute values to be examined in a form that is independent of the physical
19 storage format.

20 The decompressed intermediate format is independent of the physical
21 storage format, which may vary in different types of media. In a preferred
22 embodiment, the intermediate format is also independent of any changes to the
23 version level of the physical format. This independence allows the query logic
24 subsystem 210, which is the primary client of the physical-to-logical subsystem
25 244, to examine and understand data in this representation to resolve queries.
26 The decompressed intermediate format conversion interface is primarily record-
27 based, with provisions for conversion within parcel subdivisions, if necessary.
28 The conversion process may also use metadata tables (described below) to
29 facilitate the translation to the decompressed intermediate format from newer
30 physical storage format data specification levels.

31 The decompressed intermediate format exists to prevent the time and
32 space overhead required to maintain records in full logical data model format.

1 Entities in logical data model format are much larger in size than in the
2 corresponding decompressed intermediate format. Logical data model entities
3 are fixed-sized entities that are known to the application software and returned
4 from the query logic system 210 through the interface 212. Deferring
5 conversion from intermediate to logical data model format to only those records
6 that are returned to the navigation application minimizes the amount of memory
7 and conversion overhead required for entities that do not satisfy a particular
8 query. There is a one-to-one relationship between decompressed intermediate
9 format and logical data model entity types.

10 The transformation into the decompressed intermediate format relies on
11 an interface 257 provided by the physical-to-logical subsystem 244. This
12 utilizes the intermediate format which localizes all data corresponding to an
13 entity within the same parcel. The amount of memory required to hold the data
14 in this intermediate format is preferably smaller than the amount of data that
15 would be necessary to hold the same data entities in the logical data model
16 format. Finally, when entities that comprise the result set have been identified
17 by the query resolution logic, another physical-to-logical interface 255 is
18 available to translate the data to the logical data model format for return to the
19 navigation application programs 200.

20 **B. Cursor management**

21 The query result from the data access interface layer 41 is copied into
22 memory buffers provided by the navigation application software programs 200
23 regardless of whether or not cursors are involved. This allows the data access
24 interface layer 41 to compress (i.e. compact) the memory allocated to it in its
25 memory pool without affecting the navigation application programs 200. The
26 cursor management subsystem 249, mentioned above, is part of the query logic
27 subsystem 210. When a cursor-based query call is invoked in the data access
28 interface layer 41, an internal cursor is constructed by the cursor management
29 subsystem 249 and a unique reference to the cursor is returned to the one of the
30 navigation application programs 200 that requested the data. The cursor
31 reference is used to obtain all or some portion of the resulting data. When data

1 are obtained via a cursor, the navigation application program 200 specifies a
2 memory buffer in a size that is a multiple of the returned logical data model
3 entity size. This multiple is based on how many records at a time the
4 navigation application program prefers to (or can afford to) handle at one time.
5 In a preferred embodiment, this technique is facilitated because the logical data
6 model entities are of a known fixed size.

7 The cursor management subsystem 249 stores the entities that comprise
8 the cursor result set in two different ways. Referring to FIG. 5A, some of the
9 entities in the result set are saved in fully-translated logical data model form.
10 Whether or not all entities in the set are maintained in the logical data model
11 form depends on the number of entities that make up the query result set. For
12 cases in which the result set size is below a first threshold, the entire set is
13 maintained as an array 511 of decompressed logical data model entities. For
14 cursors whose results set exceeds this first threshold, the remaining are
15 maintained in a second array 513 that includes only entity identifiers. The
16 entity identifier (which in many cases may be the database identifier, or
17 "DBID") allows the record in the physical parcel to be quickly accessed for
18 transformation into the logical data model format.

19 The memory used to maintain the cursor result set is dynamically
20 allocated using the internal private dynamic memory management interface
21 provided by the resource management subsystem 220, as explained further
22 below. When very large result sets are encountered, a partial result set may be
23 maintained if the number of entities satisfying the query exceeds yet another
24 threshold. This second threshold is provided to minimize any possible adverse
25 memory impact of a very large result set. Each of these thresholds may be
26 configured at the time that the executable module 18 including the navigation
27 application programs 200 and the data access interface layer 41 are compiled.
28 The full or partial nature of a particular query is reported to the calling
29 navigation application program 200 so that the number of records returned by
30 the query can be properly interpreted.

31 When a partial result set is maintained for a reference cursor, the query
32 resolution process temporarily comes to a halt once the maximum partial query

1 limit is encountered. In order to continue the query at a later time, the cursor
2 also maintains enough information to resume the query resolution process at the
3 point it left off.

4 The cursor management subsystem 249 provides additional cursor
5 functionality. Once a cursor is defined for a particular query, the cursor is
6 positioned at the beginning of the result set. A "fetch next" cursor function fills
7 the result buffer of the navigation application (specified in the query call which
8 established the cursor) with the next N records and positions the cursor after the
9 last record returned. The cursor is maintained as a "window" of logical data
10 model records that moves across the result set, so that the cursor fetch process
11 is simply a memory-to-memory copy from the cursor's address space internal to
12 the data access interface layer 41 to the address space of the navigation
13 application program 200.

14 Various forms of cursor manipulation functions are also provided by the
15 cursor management subsystem 249 including the ability to reset the cursor
16 position to the beginning, end, or an absolute position in the result set. A "get
17 previous" cursor function is also provided to augment the "get next"
18 functionality. The absolute positioning function is useful in the context of the
19 result set record count returned by the query function which establishes the
20 cursor.

21 **IV. Index management and navigation (IMN) subsystem 242**

22 Referring again to FIG. 3, the index management and navigation
23 subsystem 242 contains a portion of the physical format-specific index
24 management and navigation software in the software library that forms the data
25 access interface layer 41. The index management and navigation subsystem
26 242 allows the query logic subsystem 210 and the resource management
27 subsystem 220, above and below the index management and navigation
28 subsystem 242, respectively, to be independent of the physical storage format
29 used on the medium.

30 Index information is used to resolve the location of a parcel which
31 contains a desired entity or entities based on various search criteria. Indexes

1 are also used in some cases to locate a particular data subset within a parcel.
2 (An index which is used to resolve to a particular parcel may be regarded as an
3 *external index* and an index used to resolve to a parcel internal subdivision may
4 be regarded as an *internal index* because it is stored in the parcel.) External
5 indexes are used to minimize or eliminate having to physically access parcels
6 that may not contain the requested data. Both external and internal indexes also
7 help to minimize the amount of decompression and translation that needs to
8 occur in order to inspect candidate entities. Because various aspects of working
9 with index information to resolve to the parcel, parcel data subset, or level are
10 related to the specific physical storage format, this functionality is implemented
11 within the format-specific index management and navigation subsystem 242.

12 Some of the characteristics of index information that depends on the
13 particular physical storage format include the unique identifier for the parcel
14 containing the root of the index on the physical media, what type of index is
15 used, and whether the index resolves to the parcel, data subset parcel, or record
16 level. Additionally, certain types of index information may exist on one type of
17 storage format, but not on another. In order to maintain information about
18 these index characteristics, a framework exists to maintain this information
19 within the index management and navigation subsystem 242. This information
20 may reside on the physical medium. This information may be read at
21 initialization time and maintained in memory for use by index management and
22 navigation subsystem 242.

23 This index information framework that maintains the details of the
24 indexes present on the physical storage format includes a set of interfaces 245
25 that the data query logic subsystem 210 can use to obtain the information it
26 needs to determine the optimum index to use based on the query parameters
27 received via a call from the query logic subsystem 210. In a preferred
28 embodiment, a minimum subset of indexes is provided, regardless of the
29 physical storage format. Using a minimum subset of indexes minimizes
30 unnecessary overhead in the query logic subsystem 210 that would result from a
31 completely flexible set of indexing capabilities. The indexing information
32 framework 245 also provides a structure upon the query parameter set that is

1 passed to the index management and navigation subsystem 242 to resolve a
2 query using a particular index. Finally, this framework is extensible, allowing
3 new index types to be defined for future physical storage formats, and allows
4 indexes to be defined in terms of entities and/or attributes that do not exist in
5 current database versions or specifications.

6 Several different indexing schemes are used, depending on media size,
7 media access characteristics, entity type, and the physical organization of a
8 particular physical storage format. For example, a kd-tree, or variants thereof,
9 is used for 2-dimensional resolution of spatial data, and a wide-and-shallow B-
10 tree is used for certain nonspatial data in a physical storage format used for
11 CD-ROM. The index management and navigation subsystem 242 implements
12 the logic used to locate and navigate within these different types of index trees.
13 For external index information, the leaf nodes of these trees specify a particular
14 parcel identifier. For index information that is internal to a parcel, the leaf
15 nodes instead specify a particular parcel data subset or record. Each unique
16 type of index intended for use in the data access interface layer 41 is
17 implemented within a separate module, preferably in C source code. This
18 minimizes code size by reducing unnecessary code from being included when a
19 particular indexing scheme is not used in a particular physical storage format.

20 Because external indexing information is stored in parcels on the
21 physical media, the index management and navigation subsystem 242 uses the
22 resource management subsystem 220 to physically read the index information
23 from the storage medium into a memory buffer. In alternate embodiments,
24 these index parcels may be retained in memory as the index navigation process
25 proceeds, since it can be expected that some of these indexes will be reused
26 within the same function call or even beyond it. For this reason, index parcels
27 are retained in memory by the caching logic implemented by the resource
28 management subsystem 220.

29 In one embodiment, when a task (such as a function in the query logic
30 system 210) requires an index, the index management subsystem 242 first
31 checks for the availability of the index. If the desired index is available, the
32 index management system returns a message to the task indicating availability

1 of the index and includes a handle for searching using the index. The handle
2 includes a pointer which is called to perform searches using the index. The
3 task obtains memory storage (as explained below) and constructs information
4 for the key of the index. The task also obtains memory to hold the result of
5 the index search.

6 **V. Physical-to-Logical Data Translation (P2L) Subsystem 244**

7 The physical-to-logical data translation (P2L) subsystem 244 includes
8 the storage format-specific data translation software in the software library that
9 forms the data access interface layer 41. This allows the data query logic
10 subsystem 210 and the resource management subsystem 220 to be independent
11 of the physical storage format.

12 An interface 260 for translation of an entity in the physical storage
13 format form (which may be a compressed form) to the decompressed
14 intermediate form is provided by the physical-to-logical subsystem 244. This
15 interface 260 first decompresses the entity. This is followed by a
16 metatranslation step 261, if necessary. The memory buffer used to hold the
17 result is provided by the data query logic subsystem 210. The metatranslation
18 step 261 can be bypassed when the version levels of the physical storage format
19 and the intermediate decompressed format are equal for greater efficiency.

20 Once the data query logic subsystem 210 identifies an entity to be
21 returned to the navigation application program 200, it uses an interface 263 in
22 the physical-to-logical subsystem 242 that translates the entity from the
23 decompressed intermediate form to the final logical data model form. For non-
24 cursor single-entity queries, the calling navigation application program 200
25 supplies the memory buffer which holds the translated logical data model
26 output. Otherwise, a dynamically-allocated private cursor memory buffer is
27 supplied by the data query logic subsystem 210. Unlike the table-driven
28 metatranslation process, the logical data model translation is coded into the
29 physical-to-logical software implementation.

30 It may be desired for entities in the decompressed intermediate form to
31 persist within or even across data access calls for subsequent repeated access.

1 This could minimize the CPU overhead required to repeatedly perform
2 decompression and metatranslation on the same data as well as minimize the
3 memory management overhead involved with otherwise repeatedly allocating an
4 output buffer. The management of this persistence is provided by the data
5 query logic subsystem 210. This approach takes the anticipated CPU and
6 memory savings into account, along with examination of data access patterns to
7 determine how often repeated access to the same decompressed data occurs.
8 The overall performance of the metatranslation process is also a factor in this
9 analysis.

10 In an alternative embodiment, some data entities may undergo direct
11 transformation from the physical storage format into the logical data model
12 format without intermediate translation into the decompressed intermediate
13 format.

14 Metatranslation describes the translation of data from the compressed
15 physical storage format representation at some particular data version level (also
16 referred to as a "specification level") to the intermediate decompressed
17 representation known to the data query logic subsystem 210 at some other data
18 version level. Metadata tables 259 are used to facilitate the transformation from
19 one version level to another. The physical storage format metadata tables 259
20 are read off of the medium into semipermanently allocated memory at system
21 initialization time. The metadata process is described more fully below.

22 **VI. Resource management (SRM) subsystem 220**

23 The resource management subsystem 220 provides access to memory
24 and I/O resources for the software library that forms the data access interface
25 layer 41. The resource management subsystem 220 manages the availability of
26 navigation system memory and I/O resources for use by the data access
27 interface layer 41 in a multi-tasking environment in which the navigation
28 application programs 200 also contend for these resources. In a preferred
29 embodiment, the resource management subsystem 220 does not manage
30 memory and I/O resources for the navigation application software programs 200

1 for tasks other than accessing the geographical database via the interface layer
2 41.

3 There are two primary data access interfaces to the resource
4 management subsystem 220. The first allocates and frees workspace memory
5 from a pool that is reserved exclusively for use by the data access interface
6 layer 41. The second interface specifies a parcel identifier and returns a pointer
7 to a cache memory buffer that contains the parcel. The priority of the parcel
8 can be specified, i.e. an indication of how soon (relatively) the parcel is needed.
9 This feature enables prioritization of parcel I/O transactions and persistence of
10 parcel data in cache when cache is in contention. When a requested parcel is
11 not found in cache, a physical I/O transaction is initiated to read the parcel off
12 the medium. There are additional interfaces in the resource management
13 subsystem 220 that are used to allow the navigation application software 200 to
14 adjust the resource management strategies at run time.

15 There are at least two approaches that the resource management
16 subsystem 220 uses to increase performance. The first is to minimize and
17 optimize I/O transactions within a constrained memory environment consisting
18 of a portion of the navigation system's heap memory that is dedicated to the
19 software library that forms the data access interface 41. The second is to
20 manage access to this relatively small portion of memory. Memory
21 management also includes methods for maintaining data in cache buffers to
22 minimize I/O and for compacting of the dedicated heap memory. These
23 approaches are achieved with small amounts of CPU and memory overhead. In
24 order to implement these approaches, the physical organization of data on the
25 storage medium, as well as the query optimization techniques implemented in
26 the data query logic subsystem 210 and index management and navigation
27 subsystem 242, is taken into account.

28 This resource management subsystem 220 works across a wide variety
29 of media types, physical storage formats, operating system environments, and
30 device driver capabilities. A variety of mechanisms are provided to optimize
31 the resource management subsystem 220 for a particular navigation system
32 platform and navigation software application. These mechanisms include the

1 setting of configuration parameters at compile time. Also included are a set of
2 function calls in the data access interface layer 41 that influence resource
3 management behavior at run time. Additionally, the calls for data access are
4 designed to take advantage of built-in resource management capabilities such as
5 background data access. Further, the resource management subsystem provides
6 for prioritization of requests for multiple parcels when more than one parcel is
7 requested by a function (i.e. function level granularity), and alternatively the
8 resource management subsystem can even provide for prioritization of requests
9 for multiple parcels even when parcels are requested by more than one function.

10 FIG. 4 illustrates one embodiment of the interface layer 41 used in a
11 multi-tasking environment in a navigation system. Each navigation application
12 task, APP1, APP2 ... APPn, that makes data access interface calls is linked with
13 a interface layer software library, LIB1, LIB2 ... LIBn, in shared code space.
14 Each library implements the data query logic subsystem 210, the index
15 management and navigation subsystem 242, and the physical-to-logical
16 subsystem 244, as well as most (or all) of the resource management subsystem
17 220. As shown in the embodiment of FIG. 4, the I/O manager (IOM) portion
18 270 of the resource management subsystem 220 is implemented as a separate
19 process from the rest of the subsystems (i.e. the data query logic subsystem
20 210, the index management and navigation subsystem 242, and the physical-to-
21 logical subsystem 244, as well as most of the resource management subsystem
22 220) of the data access interface layer 41. This means that the I/O manager
23 270 may not be statically-linked with the application programs 200 as are the
24 rest of the subsystems in the interface layer, but instead operates with each
25 separate navigation processes, 200A, 200B, ... 200n, via the linked portions of
26 the interface layer in each process. This allows the I/O manager 270 to
27 continue to process physical I/O in the background while these other tasks
28 continue to run. This embodiment of the I/O manager 270 would be used in
29 navigation systems that do not have a device driver that supports asynchronous
30 I/O notification capability. However, an alternative embodiment of the I/O
31 manager 270 may be used if the navigation system is of a type that includes a

1 device driver that supports an asynchronous I/O notification capability. This
2 alternative embodiment is described in more detail below.

3 **A. Memory Management**

4 As shown in FIG. 4, a portion 278 of the system heap memory 276 is
5 allocated exclusively for use by the data access interface layer 41. The portion
6 278 is the interface layer memory pool. This memory pool 278 is separate
7 from the portion 283 of the memory used by the navigation applications 200.
8 The size of the memory pool 278 is determined by the navigation application
9 program 200 through a call to the resource management subsystem 220 through
10 the data access programming interface 212. The resource management
11 subsystem 220 allows the navigation application program 200 to change the size
12 of the pool 278 dynamically at run time (as illustrated by the arrow 275
13 between the two subdivisions). In one embodiment, the minimum size of the
14 memory pool 278 is 256 Kbytes, with additional memory provided for CD-
15 ROM media to increase the amount of data in cache.

16 The memory pool 278 is used by the data access interface layer 41 for a
17 cache for parcels of varying size that are read from the physical medium as
18 well as for a general-purpose heap that is private to the software library
19 functions of the data access interface layer 41. Buffers allocated from this heap
20 memory are used as general working space to hold decompressed data records
21 and for cursor result sets.

22 The memory used for cache buffers tends to be larger and more uniform
23 in size than memory allocated from the general-purpose private heap memory,
24 which is typically smaller and more variably-sized. The management of cache
25 buffers also takes into account the fact that a strategy for persistence is
26 provided. Additionally, searching for a particular parcel in cache should incur
27 as little overhead as possible.

28 These dissimilar yet overlapping memory management objectives are
29 supported by a two-tiered approach. The first tier is responsible for the
30 management of the more uniform and larger buffers which are suitable for
31 parcel cache purposes. The second tier operates in terms of these larger buffers

1 and divides them into smaller and less uniform pieces for general-purpose heap
2 memory allocation. The cache buffer persistence strategy is implemented in
3 terms of the first tier of memory buffer management.

4 Both tiers of memory management are performed by a memory
5 management library ("MML" or "memory manager") 280. The memory
6 management library 280 is part of the locally resident portion of the resource
7 management subsystem 220 that is linked with the navigation application
8 software 200 and runs as part of each of the navigation application processes.
9 The memory management library 280 is a shared code library that uses mutual
10 exclusion technique (described below) to allow multiple applications to have
11 concurrent access to memory. For example, there may be multiple concurrent
12 navigation application processes, such as processes 200A, 200B, ... 200n. The
13 memory management library 280 also implements the cache management
14 process.

15 This two-tiered memory management architecture minimizes the amount
16 of memory required for control structures used to keep track of internal
17 memory allocation. These control structures, together with the cache
18 management control structures, are resident in the memory pool 278 of the data
19 access interface layer 41. Interprocess communication (IPC) techniques such as
20 semaphore protection are used by the memory management library to serialize
21 memory requests by arbitrating access to these control structures in a
22 multi-tasking environment. This is illustrated by the dotted lines 277
23 connecting the memory management library 280 from each navigation
24 application task, 200A-200n, to the memory pool 278.

25 Finally, a portion of the memory allocated from the memory pool 278 is
26 used by the software library components of the data access interface layer 41.
27 This portion of the memory pool is not shared (or otherwise returned) to the
28 navigation application software program 200. This is facilitated by providing
29 that the navigation application program 200 allocate its own private buffers to
30 hold returned logical data model data entities. This isolation allows the
31 resource management subsystem 220 to pursue compaction strategies to

1 minimize fragmentation of the memory pool 278 without requiring the
2 navigation application programs 200 to handle this memory management task.

3 **B. Cache management**

4 Requests for data and index parcels are routed through the memory
5 management library 280 in terms of a parcel identifier. A search for the parcel
6 in cache memory is performed. A physical I/O request to the I/O manager 270
7 occurs when the parcel cannot be found in cache. Once the parcel is in cache
8 memory, an interface 285 in the resource management subsystem 220 locks the
9 parcel in the buffer when the buffer address is returned to the data query logic
10 subsystem 210. This prevents the buffer from being swapped out when a data
11 access interface call is active. An additional interface 279 of the resource
12 management subsystem 220 releases this lock, and is called before the data
13 access interface call returns control to the navigation application program.
14 Swapping may occur when a large number of concurrent parcel requests causes
15 the system to approach memory overcommitment.

16 The persistence strategy for the cached parcels is also implemented in
17 the memory management library 280. This strategy takes buffer locks, priority,
18 usage history, and usage counts into account in order to determine which
19 buffers should be swapped out.

20 The navigation application software program 200 is in position to
21 anticipate what data may be needed eventually versus what data are required
22 immediately. In order to differentiate data access requests between immediately
23 required data and data to prepare for subsequent access, the resource
24 management subsystem 220 supports synchronous (i.e., waits for result in the
25 foreground) and asynchronous (i.e., processing continues while I/O proceeds in
26 the background) parcel requests. The asynchronous calls are used by "cache
27 prepare" data access interface functions. An asynchronous request allows the
28 navigation application program 200 to continue doing useful work while the I/O
29 transaction proceeds in the background. An interface call 281 in the resource
30 management subsystem 220 is also provided to allow the navigation application
31 program 200 to cancel a asynchronous data request. In addition, a resource

1 priority value is provided in each data access interface call to give even finer
2 control to the navigation application program 200 to manage these foreground
3 and background requests.

4 **C. I/O manager 270**

5 In the embodiment shown in FIG. 4, the I/O manager 270 receives
6 requests for physical I/O. These I/O requests originate with the synchronous
7 and asynchronous "request parcel" interfaces 287 provided by the resource
8 management subsystem 220 whenever the requested parcel cannot be found in
9 cache memory. When this occurs, the memory management library 280
10 initiates an I/O request via the I/O manager interface 269 which results in an
11 I/O transaction request being put into an I/O manager message queue. The I/O
12 manager interface 269 is depicted in FIG. 4 in the statically-linked portion 271
13 of the resource management subsystem 220. In this case, contention to the I/O
14 queue is managed using interprocess communication techniques such as
15 message queues 273 connecting the I/O manager interface 269 to the separate
16 I/O manager process 270. The message queues 273 provide a single point of
17 event notification for each process (i.e. client). The message queues 273 may
18 be used for notification of events, including I/O completion, memory allocation
19 completion, or resource access allowance. Other events may also be supported.
20 There is one client message queue for each process (i.e., client).

21 In a preferred embodiment, I/O transactions arriving via the queue 273
22 are received in terms of parcel identifiers. Also included may be an indication
23 of the priority of the request, an identification of the task (client) making the
24 request, an indication of a message queue used by the originator of the request.
25 By using parcel identifiers, the memory management library 280 and the I/O
26 manager 270 can remain independent of the physical storage format.
27 Dependencies on the physical storage format in the resource management
28 subsystem 220 are discussed below. Translation from parcel identifier to
29 physical media address occurs for these I/O requests. When the I/O transaction
30 is dispatched to the media device, the I/O manager 270 requests a cache buffer
31 from the memory management library 280 to hold the data to be read from the.

1 media. This reduces interprocess communication overhead by preventing
2 cancelled requests from needlessly requesting and releasing a buffer, and
3 minimizes cache contention by allocating the buffer at the latest possible time.

4 There are two functions that the I/O manager 270 provides to enhance
5 system performance. One function already mentioned is to allow physical I/O
6 to occur in parallel with other activities. This allows other navigation
7 application software functions 200 or data access interface layer software 41 to
8 continue to run while a physical I/O transaction is in progress. The other
9 function provided by the I/O manager 270 is a serialization function that allows
10 incoming I/O requests to be reordered. This reordering is based on several
11 factors, including the priority of the request, the physical location of the data on
12 the media, the current read head position. Other factors may also be included.
13 Note that even within a single transaction submitted by a single navigation
14 application process, multiple parcels may be specified. This means that I/O
15 reordering can result in increased performance even when only a single task is
16 requesting data (i.e., "functional level granularity"). Additionally, because
17 index and data information may appear redundantly, particularly for physical
18 storage formats for media with relatively slow random access performance such
19 as CD-ROM, the I/O manager 270 selects the closest of the redundant parcels
20 to read. This feature is provided in part by the support for scattered reads
21 provided by the I/O manager 270.

22 All of the reordering capabilities described above use the ability of the
23 I/O manager 270 to maintain the current physical read head location. This is
24 obtained from the media device driver 290 (FIG. 3). The I/O manager 270 also
25 depends on the ability to obtain a physical media address (or addresses, for
26 redundantly-placed parcels) from an "opaque" parcel identifier. (By "opaque" is
27 meant that the I/O manager 270 can pass along the parcel identifier without
28 knowing to what it specifically refers on the physical storage format medium.)
29 The physical medium address depends on the physical storage format. These
30 dependencies are isolated from the generic portion of the I/O manager 270 by
31 two additional resource management components: a physical storage format

1 address mapper (PAM) 296 and a file directory mapper (FDM) 298. Media
2 device isolation is provided by a media device isolation layer (MDIL) 300.

3 (In a preferred embodiment, the I/O manager 270 handles all data read
4 from the medium. Under certain circumstances, the navigation application may
5 access the data on the medium directly. For example, if there is a specific type
6 of third party data stored on the medium, it may be preferable for the
7 navigation application to access it directly. In addition, certain other types of
8 data may benefit from being read directly from the medium or bypassing
9 storage in cache. These types of data include sound or video data. Due to the
10 way these types of data are used, it may be preferable for sound or video data
11 to bypass cache storage and stream directly to the navigation application for
12 presentation to the end-user.)

13 **D. File directory mapper 298**

14 The geographic database 40 exists in one or more physical binary files
15 on the storage medium 22. The parcel identifiers for all data in these files,
16 both index and map data information, are closely related to a physical offset or
17 address in one of these files. All interface layer I/O from the I/O manager 270
18 is performed in terms of a physical media address and an extent (size). These
19 addresses and extents are represented in terms of an integer number of physical
20 subdivisions on the medium. For CD-ROM media, this physical subdivision is
21 the 2 Kbyte sector. The I/O manager 270 uses the services provided by the
22 physical storage format address mapper 296 to obtain the physical media
23 address for a parcel relative to the beginning of its file. In order for the I/O
24 manager 270 to generate the absolute media address, the starting location of the
25 file is determined. This information is provided by file directory mapper
26 (FDM) 298. A file directory exists at some known location on the medium.
27 This directory provides a physical media address for each of the files on the
28 medium, regardless of whether they are geographic files or not. This file
29 directory is specified for the particular medium. In a preferred embodiment,
30 the ISO-9660 standard filesystem for CD-ROM media is used, although other
31 alternative file systems may also be suitable. For read/write media, a DOS

1 FAT file system may be provided, with the geographic file stored as a read-
2 only file organized contiguously on the medium to bypass the FAT indirection.

3 The file directory mapper 298 isolates the I/O manager 270 from the
4 media-specific (and therefore physical storage format specific) file directory
5 organization. The file directory mapper 298 provides an interface which
6 translates a file name to the physical media address of the beginning of the file.
7 This interface may be available to the navigation application software 200, so
8 that the locations of third party or navigation application-specific files, if any,
9 can be obtained. This interface allows the navigation application to implement
10 a file I/O framework based on file descriptors and byte offsets so that third
11 party or navigation application-specific files can be accessed via the media
12 device driver interface.

13 The file directory mapper 298 is able to ascertain the location of the file
14 directory on the medium 22 based upon a predetermined information of its
15 structure. When invoked, the file directory mapper 298 reads the directory
16 from the known location on the medium. This is illustrated by the interface
17 301 between the file directory mapper 298 and the media device 22. A private
18 buffer obtained from the memory management library 280 is used to hold a
19 temporary copy of the directory until the offset is determined. This activity
20 takes place at initialization time for all geographic data files to minimize
21 repeated access of the directory. The starting location for each file is retained
22 within the file directory mapper 298 for the duration of an operating session.

23 **E. Physical storage format address mapper 296**

24 As mentioned above, the physical storage format address mapper 296
25 translates a parcel identifier (i.e. a "parcel ID") into a physical media address
26 (or addresses) relative to the beginning of the file the parcel is located in.
27 More specifically, the physical storage format address mapper 296 translates a
28 parcel ID to a physical sector address and sector count on the medium by using
29 the file address mapper 298 to locate the file on the medium and translating the
30 logical parcel ID to a starting physical sector and sector count. Multiple
31 addresses are returned for redundantly placed index and data parcels. The

1 address mapper 296 is a component of the I/O manager subsystem 270 that
2 does not physically interact with the medium 22.

3 The parcel identifiers are closely related to the physical addresses of the
4 parcel on the medium. The identifier also carries information about parcel size
5 and a bit (or other information) which indicates the presence of redundant
6 copies of the parcel on the medium. An example of a parcel ID 600 is
7 illustrated in FIG. 6. The parcel ID 600 may consist of one part 601 that is an
8 offset from the start of the file containing the parcel, a second part 602 that
9 indicates the size of the parcel, and a third part 603 that indicates whether there
10 are redundant copies of the parcel on the medium. The locations of redundant
11 copies of the parcel are determined by use of a table which is retained in global
12 memory throughout an operating session of the navigation system. This
13 information can be combined algorithmically within the physical storage format
14 address mapper 296 in a physical storage format-specific way to generate the
15 physical address using minimal CPU or memory table overhead.

16 The physical media address (or addresses for redundantly stored parcels)
17 returned by the physical storage format address mapper 296 is not a byte
18 address, but is instead at a physical subdivision of the medium. For CD-ROM
19 media, this subdivision is the 2 Kbyte sector. For read/write media, the
20 subdivision may be a smaller size such as 256. Similarly, the parcel size may
21 also be expressed in these terms.

22 The interface to the physical storage format address mapper 296 takes a
23 parcel identifier (parcel ID) and returns a relative physical media address (or a
24 set of physical media addresses for redundantly stored parcels) along with the
25 parcel size. This information corresponds to the portions 601 and 602,
26 respectively, of the parcel ID. The relative physical media address (or
27 addresses) is added to an absolute file starting location address that has been
28 obtained from the file directory mapper 298 at initialization time to produce an
29 absolute physical address (or addresses) for the parcel. If redundant copies of
30 parcel are provided, the I/O manager 270 then chooses the closest address based
31 on current read head location, and passes this address, the parcel extent (size),

1 and the address of the cache buffer obtained from the memory management
2 library 280 to the media device isolation layer 300 to initiate the physical I/O.

3 **F. Operating system isolation layer (OSIL) 302**

4 An operating system isolation layer (OSIL) 302 provides an interface
5 between the generic operating system services to which the data access interface
6 layer 41 is written and the particular services provided by the operating system
7 202 of the navigation system platform. For example, the operating system
8 isolation layer 302 can be utilized to support various operating systems
9 including ITRON, OS9, pSOS, VRTX, VxWorks, proprietary operating
10 systems, and single task operating systems. The operating system isolation
11 layer 302 is a software module that is related to a specific platform (or OS) and
12 enables the data access interface layer 41 to operate on a particular platform.

13 There are several different types of operating system services. The first
14 consists of general purpose services, such as string functions (including
15 transformation between multi-byte and wide-character formats), math functions,
16 common utilities such as searching and sorting, and the like. The software
17 library that comprises the data access interface layer 41 uses some of these
18 operating system services. For example, an ANSI C standard library (*stdlib*)
19 interface is used for these functions. The memory management functions
20 *malloc()* and *free()* also fall under the *stdlib* interface. These are used by the
21 resource management subsystem 220 to allocate and resize the memory pool
22 278 for internal management (as mentioned above).

23 The other set of OS-level interfaces that the resource management
24 subsystem 220 uses includes the protection of shared data structures in memory
25 such as the memory management and cache control structures, and the I/O
26 queue. Contention to these structures by multiple tasks linked to the data
27 access interface layer 41 is managed using interprocess communication
28 mechanisms such as semaphores and message queues. The resource
29 management subsystem 220 preferably uses the standard POSIX.4 interfaces for
30 these mechanisms.

1 If another type of interface is provided by the operating system 202 of
2 the navigation system 10, the operating system isolation layer 302 implements
3 the ANSI C *stdlib* and POSIX.4 interfaces as a translation layer to the
4 interfaces for the native services provided by the navigation system OS. If it is
5 not, then the service is implemented within the operating system isolation layer
6 302.

7 Note that even though the resource management subsystem 220 uses
8 logical POSIX.4 semaphore and message queue interfaces, this does not
9 necessarily mean that a physical semaphore or message queue is used
10 underneath. For example, it may be faster and/or easier, compared to using a
11 physical semaphore, to simply disable interrupts while in a critical section of
12 code, such as reading or updating a shared cache control structure, and reenable
13 the interrupts afterward. In a preferred embodiment, use of interrupts is limited
14 to actions that are predictably short in order to limit any impact on other tasks.
15 Alternatively, the I/O queue can be implemented as a "semaphore-protected"
16 shared memory buffer instead of a message queue when the I/O manager 270 is
17 implemented completely within the statically-linked software library of the data
18 access interface layer 41. This alternative embodiment is discussed below. A
19 still further alternative is to provide a single interface layer task with no
20 multi-tasking contention to these internal resource management structures
21 whatsoever.

22 The operating system isolation layer 302 allows a definition for the
23 standard interfaces to which the data access interface software library 41 can be
24 written. Some of the operating systems to which the data access interface layer
25 41 may be ported provide some or all of these interfaces. In these cases, the
26 interface service provided by the OS can be used instead of the corresponding
27 function provided by the operating system isolation layer 302. Some UNIX
28 variants and some embedded-oriented operating systems provide complete ANSI
29 C *stdlib* and POSIX.4 services.

30 **G. Media device isolation layer (MDIL) 300**

1 The media device isolation layer (MDIL) 300 serves a similar function
2 to the operating system isolation layer 302, except for the media device driver
3 interface. In general, particularly for slow CD-ROM media, performance is
4 improved significantly when scattered or skipped read support is provided by
5 the device driver. This feature provides for reading some number of
6 noncontiguous but nearby sectors into the same number of separate memory
7 buffers provided by the caller application. This feature provides approximately
8 sequential-access performance without having to dispatch each request
9 separately. This feature may incur some additional rotational latency and
10 possibly require a seek backwards when used on media such as CD-ROM,
11 because data are physically stored in a spiral on such media. Additionally, this
12 feature may require having to allocate a larger contiguous memory buffer to
13 hold the desired sectors as well as any unwanted gaps.

14 The media device isolation layer 300 supports a scattered read device
15 capability without requiring the actual device to support scattered reading. The
16 I/O manager component 270 of the resource management 220 writes to a single
17 device interface which takes an array representing a sequence of physical media
18 addresses, extents, and memory buffer pointers. The size of this sequence is
19 configurable. If the media device provided by the navigation system accepts
20 this kind of scattered read interface, then the media device isolation layer 300
21 may not be needed. However, the device interface provided by the navigation
22 system 10 may not provide for this feature even if scattered read support is
23 provided. For example, a linked-list containing this information, rather than an
24 array and size may be required. Alternatively, the data structure may be
25 ordered differently. Accordingly, in a present embodiment, this type of
26 translation can be handled within the media device isolation layer 300.

27 There are also media device drivers that do not support scattered read.
28 This may be the case for fast drives or for read/write media. Alternatively,
29 relatively slow device performance may be balanced with more memory for
30 cache. In these cases, the media device isolation layer 300 dispatches each of
31 the I/O transactions into the input array one-by-one.

1 Accordingly, the media device isolation layer 300, like the operating
2 system isolation layer 302, is tailored to a specific type of media and may be
3 specifically prepared for the process of porting the data access interface layer
4 41 to a particular platform. In a preferred embodiment, the media device
5 isolation layer 300 and the operating system isolation layer 302 are the only
6 two components that are platform dependent. Further in the preferred
7 embodiment, the remaining library functions that comprise the data access
8 interface 41 have identical source code for all platforms.

9 **VII. Operation**

10 *Memory Pool Initialization and Allocation*

11 In a preferred embodiment, the amount of memory available to the
12 interface layer is, in part, a function of the particular hardware platform of the
13 navigation system. Further, in a preferred embodiment, the amount of memory
14 that is made available to the interface layer may be controlled, to a limited
15 extent, by the navigation application. Referring to FIGS 3 and 5, in one
16 embodiment, the navigation application 200 may use an API (application
17 programming interface) call to the memory manager 280 at system
18 initialization. At this point, the memory manager 280 can allocate a certain
19 amount of memory for use by the interface layer. One way by which the
20 memory manager can allocate memory for use by the interface layer is to use a
21 function call, such as *malloc()*, to the operating system of the navigation
22 application to obtain an allocation of memory.

23 The memory allocated by the memory manager 280 at initialization
24 forms the interface layer memory pool 278. The memory manager 280
25 subdivides the memory in this memory pool 278 into multiple memory blocks.
26 In a preferred embodiment, each block is contiguous area of memory of a
27 predetermined, fixed size. The predetermined memory block size may be 1
28 Kbyte or 2 Kbyte or any other size consistent with the hardware and software
29 of the navigation system. In one embodiment, the memory manager 280 first
30 obtains and allocates a minimum amount of memory required for the interface
31 layer and then obtains additional blocks of memory, by again making a function

1 call (e.g. *malloc()*) to the operating system, to obtain additional memory until it
2 has allocated all the memory that the navigation application has configured for
3 the interface layer. By allocating a single area of memory for minimum usage
4 at initialization time and then allocating additional memory later, memory
5 fragmentation can be reduced as later deallocation and reallocation results in
6 resizing the memory usage of the interface layer, as explained below.

7 *Deallocation and reallocation*

8 In a present embodiment, the interface layer advantageously provides for
9 dynamic adjustment of the amount of memory used by the interface layer. This
10 allows a portion of the memory in the navigation system to be used alternately
11 by either the interface layer or the navigation application programs depending
12 upon which of these has the greater need for the memory. In a preferred
13 embodiment, one or more of the additional blocks of memory over the
14 minimum amount allocated by the interface layer may be subsequently returned
15 to the navigation application when the navigation application requires additional
16 memory. According to this embodiment, during operation, the navigation
17 application program 200 may request that an amount of memory be returned
18 from the interface layer. The navigation application may require this additional
19 memory for a memory intensive task, such as the display of a large number of
20 items. Upon receiving the request, the memory manager 280 checks whether
21 there is additional memory allocated to the interface layer in the memory pool
22 278 over the required minimum. If so, the memory manager 280 identifies an
23 amount equal to one or more blocks, rounded up to the next higher whole block
24 over the amount requested by the navigation system. (In a preferred
25 embodiment, when allocating or deallocating memory from the navigation
26 system, the memory manager 280 handles whole blocks.) By returning an
27 amount rounded up to the next higher whole block over that amount that was
28 requested by the navigation application, the interface layer assures that the
29 navigation application receives at least the amount of memory it needs. The
30 memory manager returns the memory to the navigation application by using a
31 function call to the navigation system operating system, such as *free()*.

1 Similarly, when the navigation application program no longer requires
2 the additional memory, it can return the memory to the interface layer.
3 Returning memory from the navigation application to the interface layer may
4 follow a process similar to the initial memory allocation procedure, described
5 above. In this manner, the interface layer can be assured of a minimum amount
6 of memory for its own use, as well an additional amount of memory that it can
7 routinely use thereby providing a relatively high overall level of performance.
8 Furthermore, when the navigation application has an intensive task, it can
9 temporarily use some or all of the interface layer's memory allocation over the
10 minimum amount. This feature of the interface layer provides for an
11 advantageous use of the resources of the navigation system by allowing for
12 dynamic adjustment of the memory usage.

13 *Memory pool usage*

14 The memory manager 280 provides memory from the memory pool 278
15 for two different kinds of tasks. First, memory in the pool is used for a private
16 workspace for the use of the interface layer subsystems. Second, memory in
17 the pool is used for cache, i.e. to hold data read from the medium. When a
18 parcel of data is read from the medium, it is stored in a cache buffer. The
19 memory manager 280 keeps track of the number of available memory blocks in
20 the memory pool and the number of blocks which may be allocated for private
21 workspace. In some embodiments, newly allocated or freed blocks are
22 preferably used for cache rather than for private workspace.

23 The memory manager 280 allocates memory from the memory pool 278
24 for both cache and for private workspace in response to requests from tasks for
25 buffers. A requested buffer may be smaller, larger, or the same size as one of
26 the memory blocks in the memory pool. When attempting to satisfy a request
27 for a memory buffer larger in size than the fixed memory block size, the
28 memory manager attempts to locate a set of contiguous memory blocks that
29 satisfy the buffer size requested, and allocate these multiple contiguous blocks
30 to service the request. Requests for memory may also be made for amounts
31 less than the size of a whole block. The memory manager subdivides a

1 memory block to satisfy smaller requests. Requests for smaller amounts of
2 memory are more typically made for private workspace rather than for cache.

3 Each allocated buffer, whether for cache or private workspace, is
4 assigned a buffer ID. This buffer ID is a constant handle associated with the
5 buffer over the lifetime of the buffer. Memory pointers to the buffer may
6 change due to memory compaction. In a preferred embodiment, for buffers
7 used for cache, the parcel ID is used as the buffer ID. For buffers used for
8 internal workspace, a unique buffer identifier is generated.

9 *Task buffer counts and task buffer status*

10 The resource management system keeps track of each task's cache buffer
11 count. A task's buffer count corresponds to how many buffers are "owned" by
12 a task. Buffers are owned by a task whenever a task requests or reads data in a
13 buffer or whenever a buffer is shared with another task that had ownership and
14 the other task subsequently releases its ownership. If the buffer is used for
15 cache, the data in the buffer include a parcel read from the medium and the
16 ownership of the buffer is determined by the task that requested the parcel.
17 When more than one task requests data in a buffer, the memory manager
18 assigns ownership of the buffer to the last task that requested the data in the
19 buffer or the last task that read data from the buffer. A task loses ownership of
20 a buffer when another task takes ownership of the buffer or when the task
21 ignores the data (i.e. the parcel) in the buffer. A task also loses ownership of a
22 buffer when the buffer is swapped out. A task's cache buffer count changes
23 each time the task takes control of or loses a buffer.

24 Tasks may have assigned minimum and maximum cache buffer limits.
25 These limits relate to how many buffers can be "owned" by a task. These
26 limits may be enforced only when there is contention for available memory
27 resources. Otherwise, each task is permitted to use as much memory as is
28 available. When memory resources are in contention, tasks that exceed their
29 maximum buffer limits will lose buffers. Tasks will not lose buffers if they are
30 only at their minimum buffer limits.

1 Each buffer has a status. The memory manager 280 keeps track of the
2 status of each of the buffers. The status of a buffer derives from the status of
3 the tasks using the buffer. More than one task may be using a single buffer.
4 The memory manager 280 maintains a status of each task that accesses the
5 parcel.

6 A buffer's status may be "locked", "active", or "ignore." When the
7 contents of the buffer are currently in use by a task, the task status and the
8 buffer status are set to "locked." If any one task using a parcel in a buffer is
9 locked, then the buffer is locked as well. When a buffer is locked, it is not
10 available for swapping out of cache. Also, in a preferred embodiment, a locked
11 buffer may not be moved, such as during memory compaction.

12 A buffer in "active" status indicates that one or more tasks may still be
13 interested in the contents of the buffer, but that the buffer can be swapped out
14 if the memory in it is needed. Information about the task, its resource priority,
15 and so on, remain associated with the buffer so that these factors can be
16 evaluated by the memory manager when the manager is seeking memory to
17 swap out. Unless the buffer is needed for other tasks, the data remains in the
18 cache buffer and the buffer is available, if needed, for other tasks that may need
19 to access the data in the buffer.

20 A buffer in "ignore" status indicates that no tasks are currently using the
21 memory in the buffer. Therefore, this buffer may be used for other tasks or
22 swapped out. Ignored buffers may also be moved and/or compacted.

23 *Compaction*

24 The resource manager also provides compaction and so-called "garbage
25 collection" of the interface layer memory pool 278. As memory blocks are
26 allocated and deallocated, fragmentation may occur within the memory pool.
27 The memory manager 280 groups together adjacent unused memory blocks and
28 memory blocks having "ignore" status for subsequent reuse for cache or private
29 workspace. In addition the memory manager may physically move unlocked
30 buffers in order to decrease fragmentation.

1 In determining when to swap out or compact memory, the memory
2 manager takes into account resource priority, task buffer counts, and buffer
3 status. In general, a high priority application or an application that frequently
4 accesses its cache will have a reduced likelihood of having its buffers swapped
5 out.

6 When deallocating or compacting the memory, it may be preferred to
7 select the memory in the region closest to the original boundary between the
8 interface layer's allocated memory 278 and the application's allocated memory
9 283.

10 *Processing of a query task*

11 As mentioned above, the interface layer defines a set of calls that can be
12 used by a navigation application program to access a set of geographic data
13 entities on a physical storage medium. The set returned by the interface layer
14 can be empty, can include a single entity, or can return a list of multiple
15 entities. These returned entities conform to the logical data model, described
16 above, which is a decompressed format. The interface layer accepts calls that
17 allow the navigation application to formulate a query for data. These calls may
18 be qualified by spatial as well as non-spatial attributes of the desired entities.
19 The attributes used to qualify a query can be used singly or in combination, and
20 can specify a range or a single value.

21 In a preferred embodiment, the geographic data on the medium are
22 organized into one or more files. The data include spatially organized data,
23 non-spatially organized data, and index data. The storage medium may include
24 other types of data as well. The spatially organized data include data such as
25 segments of roads and nodes. In a preferred embodiment, these data are
26 organized into parcels which contain the data entities encompassed within
27 physical geographical localities within the geographic region. Non-spatially
28 organized data may include data, such as names of navigable features. These
29 data may be organized in a manner that facilitates their use, such as
30 alphabetically or by municipality. Indexing data include indexes that relate the
31 various spatial and non-spatial data to each other. These indexes may include

1 kd-tree indexes, B-tree indexes, and so on. In a preferred embodiment, all
2 these types of data are included in parcels, each identified by a unique parcel
3 ID. In a preferred embodiment, the parcel size is related to a physical sector
4 size on the physical medium. For example, for CD-ROM media, the sector size
5 is 2 Kbytes and the parcel size is a multiple of the sector size, e.g. 16 Kbytes.
6 In a preferred embodiment, the interface layer accesses data from the medium
7 in whole parcels and loads whole parcels into memory in a single physical read
8 in order to resolve a query. Multiple parcels may be read into memory in order
9 to resolve a query transaction.

10 When the query logic subsystem 210 receives a call for geographic data
11 from the navigation application program 200, the request may be in the form of
12 a "geo-coded" request. This means that the navigation application 200 wants
13 data relating to certain geographic conditions or boundaries. The query logic
14 subsystem 210 resolves the request into parcel ID's by mapping the request to a
15 set of parcels so that the data can be retrieved from the medium. This is done
16 using the index management subsystem 242. Accordingly, it may necessary to
17 first access the index data in order to identify the parcel ID's of the geographic
18 data needed to respond to the query. Commonly-used index data may be read
19 into memory at initialization time. If the desired index data are not already in
20 memory, they are read from the medium.

21 Once the parcel ID of the desired data is identified, the request for the
22 parcel is directed to the memory manager 280. The memory manager examines
23 the cache portion of the memory pool to determine whether the requested parcel
24 has already been stored in cache memory in response to a request from a
25 previous task. If the requested parcel is already in cache, the memory manager
26 280 returns a pointer to the cache buffer to the requesting task and locks the
27 buffer (as explained above). If the requested parcel is not already in cache, the
28 memory manager 280 sends the request to the I/O manager 270. In the I/O
29 manager, the requested parcel is compared to a list of parcels that have already
30 been requested but not yet been sent to the media driver. If the parcel is on the
31 list of requested parcels, the task's (client's) identification is added to the
32 information associated with the parcel. If the parcel is not in the list, the parcel

1 is added to the list along with an identification of the task (client) requesting
2 the parcel and information indicating the parcel's sector location and length.
3 When a task requires a plurality of parcels, the listing of parcels are sent as a
4 group to the I/O manager for retrieval in order to satisfy the request at the same
5 time.

6 For each parcel requested, the I/O manager 270 calls the physical
7 address mapper 296. The physical address mapper 296 translates the parcel ID
8 and returns the parcel size and physical starting sector (or sectors if the parcel
9 is stored redundantly). Using this information, the I/O manager 270 sorts the
10 I/O requests in priority and sector proximity order taking into account the
11 current read head position. For parcels stored redundantly, the I/O manager
12 270 selects the sector that results in the minimum search time.

13 The I/O manager 270 calls the memory manager 280 to allocate cache.
14 The memory manager 280 scans a list of free memory in the memory pool 278
15 to find available memory and adds the available memory to cache. The
16 memory manager 280 locks a buffer, as described above, to indicate that the
17 memory associated with the buffer is reserved for I/O. Data are also stored that
18 identifies the client task that requested the I/O.

19 If the memory manager 280 finds no available memory, it attempts to
20 swap out memory. As mentioned above, locked buffers are not swapped out.
21 If there are any buffers whose status is "ignore", these are swapped out first. If
22 memory is still needed in order to allocate cache for a new I/O, the list of tasks
23 is examined to see if any tasks have exceeded their maximum buffer limit. If
24 so, the memory manager 280 searches for any unlocked buffers used by the
25 task and these are selected for swapping out. If memory is still needed for
26 allocating to new data, buffers whose status is "active" are examined and these
27 are swapped out for use for the new data.

28 If the request for a memory buffer for the new I/O cannot be satisfied
29 with the available memory in the memory pool 278, the memory manager 280
30 may perform compaction of the memory pool. As described above, only
31 unlocked buffers are available for compaction.

1 It may occur that even after compaction no memory is available in the
2 size requested by the I/O manager for the new I/O. If this occurs, the memory
3 manager 280 returns information to the I/O manager 270 indicating the largest
4 cache space that is available. The I/O manager 270 then scans the I/O request
5 list to determine whether any I/O request waiting to be serviced can be satisfied
6 by the available cache. If there is an I/O request waiting to be serviced that
7 can be satisfied with the available cache size in memory, this I/O request is
8 serviced. The I/O manager 270 sends a message to the memory manager 280
9 to lock the cache memory that is available in the requested size. The I/O
10 manager 270 then sends the I/O request to the media driver for retrieval.

11 If no I/O request in the I/O request list matches the available memory,
12 the I/O manager 270 temporarily stops generating new requests for cache to the
13 memory manager 280. Eventually, as tasks are completed, the memory
14 manager 280 unlocks or frees memory and as memory becomes available, the
15 pending I/O requests are serviced.

16 When the memory manager 280 indicates to the I/O manager 270 that
17 cache is available to service an I/O request, the buffer is reserved. The I/O
18 manager services the I/O request list by building scattered read requests. A
19 number of reads may be chained together. Once this information is sent to the
20 media driver, the I/O requests are removed from the I/O request list.

21 The memory manager 280 sets the status of the cache buffer to "active."
22 The memory manager returns a pointer to the buffer back to the task that
23 requested the data, e.g. a call from the query logic subsystem. Once the pointer
24 to the buffer is returned to the task that requested it, the buffer's status is set to
25 "locked." The data in the cache buffer are used as appropriate. The pointer to
26 the buffer is not changed until the task unlocks the buffer, e.g. by indicating
27 that it has finished with the data in the buffer. When the data (in the logical
28 data model format) are returned to the navigation application and therefore the
29 data in the cache buffer are no longer needed by the task, the task in the query
30 logic subsystem 210 also returns a message to the memory manager to reset the
31 cache buffer from "locked" to "active" or "ignore" indicating that the data are

1 not presently needed. When it is no longer "locked", the buffer may be
2 compacted, moved or released.

3 In general data are maintained in cache for as long as possible consistent
4 with other requirements. This provides the advantage that once data are in
5 memory they are stored in a cache because of the possibility that the data may
6 be needed for a subsequent task. Therefore, the data may persist in cache over
7 a period of time that includes several calls from the navigation application. In
8 order to provide for memory compaction and defragmentation, data which are
9 not locked may be moved by the memory manager 280 from one memory
10 location to another. In order to find the data which may be in cache, a task
11 requests the data by the buffer ID, which in a preferred embodiment is the same
12 as the parcel ID for cache data. Therefore, when a task requests a parcel, it
13 obtains a pointer to an address of a buffer. The "pointed-to" buffer has a buffer
14 ID which is the same as the requested parcel ID. In this manner, the memory
15 manager 280, by keeping track of buffer ID's and their addresses, returns a
16 pointer to the task indicating where the buffer is located. If the memory
17 manager 280 moves a buffer during compaction, it keeps track of new location
18 of the buffer so that the appropriate pointer can be returned to any task
19 requesting that particular buffer. Pointers maintained in memory buffer itself
20 are by offset and not absolute address as the base block address can move.

21 *Processing of a request for private workspace task*

22 As mentioned above, the memory manager 280 also allocates buffers for
23 private workspace for interface layer functions. As mentioned above, if the
24 memory required by an interface layer function is smaller than one of the fixed
25 sized blocks in the memory pool 278, the memory manager 280 subdivides the
26 block into smaller sized portions. When a request for private workspace is
27 made, the memory manager 280 first searches memory blocks that are already
28 in use by private memory functions to determine whether there is sufficient
29 space available in the block to meet the memory needs of the new request.
30 Additional blocks may be allocated for private workspace, if needed.

1 The memory manager 280 assigns a unique buffer ID to each block used
2 for private workspace. These buffers are locked while in use by the interface
3 layer function.

4 *Operation of the physical-to-logical data translation subsystem*

5 The physical-to-logical subsystem 244 subsystem is responsible for
6 translation of information from the optimized physical storage format found in
7 parcels read from the medium into memory by the resource management
8 subsystem 220. This translation occurs in two phases. The first phase is from
9 the compressed physical storage format to the decompressed intermediate
10 format (DIF) that can be examined to resolve queries. The second phase is
11 from the decompressed intermediate format to the final logical data model
12 format that is returned to the application via the query logic system 210.

13 *Operation of cursor management system*

14 The tasks that request data, as described above, may result in either
15 small or large amounts of data being found. The cursor management subsystem
16 249, which is part of the query logic subsystem 210, handles these results
17 regardless of the amount of data that results from a request. Operation of the
18 cursor subsystem begins with a function in the query logic subsystem that
19 expects to require use of the cursor subsystem in connection with a request for
20 data. Several different functions in the query logic subsystem 210 can expect
21 to retrieve data that requires use of the cursor management subsystem. First,
22 the function in the query logic subsystem initializes memory, i.e. a buffer, for
23 the cursor. The cursor cache is a portion of memory that holds an array of data
24 structures. The structures are determined by the query function based on the
25 query type. Information about the structures are passed to the cursor
26 management subsystem. The information may include the number of records to
27 be used in a cursor cache, record size, number of data entity ID's (DBID's) that
28 could be stored in the cursor memory, and so on. Some of this information
29 may be fixed in size, such as the DBID. Referring to FIG. 5A, this information

1 is used to set up the arrays 511 and 513 for holding the decompressed data
2 entities and the data entity ID's, respectively.

3 After the cursor's cache is allocated, the query is processed to obtain the
4 data results, as described above. If multiple entities are located that satisfy the
5 request, each of the data entities retrieved is stored in fully decompressed
6 logical data format in the array 511 until the array is full. The data entity ID's
7 (DBID's or database identifiers) for the records stored in the array 511 are
8 included in the second array 513. In addition, if the number of data entities
9 identified as matching the query exceeds the number of data entities that can be
10 stored in decompressed form in the first array 511, the data entity ID's for these
11 additional data entities are stored in the second array 513 after the ID's of the
12 data entities stored in the first array. In the example of FIG. 5A, twenty data
13 entities are stored in fully decompressed form in the first array 511 and the
14 entity ID's of 100 data entities (including the first twenty) are stored in the
15 second array 513. Any of these data entities in the first array 511 can be
16 immediately returned to the navigation application program since they are fully
17 decompressed. Both the number of entities that can be stored in the
18 decompressed array 511 and the number of database ID's that can be stored in
19 the array 513 are determined by the query function that calls the cursor
20 management system.

21 In a preferred embodiment, the first array 511 supports both FIFO (first-in-first-out) and LIFO (last-in-first-out) during use of the cursor cache. If a
22 record beyond the last record in the array is requested, for example record 21,
23 that data entity is found (using the entity's DBID in the second array 513),
24 decompressed into the logical data format, and stored in decompressed form in
25 the first array 511 replacing the oldest record (e.g. record 1). Similarly, if
26 record 22 is requested, it is found using the data entity ID from the second
27 array 513, decompressed into logical data format, and stored replacing record 2
28 in the first array 511. This type of access is in the forward direction and would
29 use FIFO. On the other hand, if record 1 were then requested, record 1 would
30 be retrieved, decompressed, and stored in decompressed form in the first array
31 511 replacing record 20. Record 20 would be replaced since it is the oldest
32

1 record in the first array 511. This type of access is in the reverse direction and.
2 would use LIFO.

3 Another example is shown in FIG. 5B. In FIG. 5B, the first array 511
4 is shown as being occupied with records 25 through 44. If record 24 were
5 requested, it would be decompressed and would be stored in the first array 511
6 replacing record 44. If record 23 were then requested, it would be
7 decompressed and would be stored in the array 511 replacing record 44, and so
8 on.

9 In FIGS. 5A and 5B, the second array 513 (i.e. the array that holds only
10 the data entity ID's) is shown to have a size that accommodate only 100 data
11 entity ID's. It is possible that the query will result in more than 100 entities.
12 When there are more entity ID's than will fit into the second array 513, the
13 cursor subsystem retains information that provides for one or more additional
14 pages of data entity ID's.

15 The following example illustrates the paging function of cursor system.
16 For purposes of this example, it is assumed that the first array 511 can hold 20
17 decompressed data entities and the second array 513 can hold 100 data entity
18 ID's, but that the query results in 1000 records.

| 1 | Cursor Page | ID's range in ID array 513 | ID's range in cache array 511 | Page header |
|----|-------------|----------------------------|-------------------------------|-------------|
| 2 | 1 | 1-100 | 1-20 | 1 |
| 3 | 2 | 80-180 | 100-120 | 80 |
| 4 | 3 | 160-260 | 180-200 | 160 |
| 5 | 4 | 240-340 | 260-280 | 240 |
| 6 | 5 | 320-420 | 340-360 | 320 |
| 7 | 6 | 400-500 | 420-440 | 400 |
| 8 | 7 | 480-580 | 500-520 | 480 |
| 9 | 8 | 560-660 | 580-600 | 560 |
| 10 | 9 | 640-740 | 660-680 | 640 |
| 11 | 10 | 720-820 | 740-760 | 720 |
| 12 | 11 | 800-900 | 820-840 | 800 |
| 13 | 12 | 880-980 | 900-920 | 880 |
| 14 | 13 | 960-1000 | 980-1000 | 960 |

15 In the example, initially the first array 511 includes records 80-100 in
 16 decompressed form and the second array 513 is includes the first 100 data
 17 entity ID's. At this point, neither the first nor the second array includes any
 18 data entities or ID's beyond record 100. If the application want to access the
 19 101th record, the query has to be resolved again. All the data entity ID's in the
 20 second array 513 are cleared and replaced with entity ID's for records 80-180.
 21 The record 101 is decompressed and stored in the first array 511 replacing
 22 record 80. The table shows the ranges for each page when there are 1000
 23 records and the arrays 511 and 513 hold 20 records and 100 entity ID's,
 24 respectively.

25 The arrays described above can be used with various types of data
 26 including types of data that do not have their own entity ID's. These types of
 27 data may be attributes associated with, or describe, entities in the database.
 28 Examples of these types of data include blocks, landmarks, and text. For these
 29 types of data, a virtual ID is assigned by the cursor management system. The

1 virtual ID would be similar to a regular data entity ID except that instead of
2 representing a type of entity, it would represent an offset, or other type of
3 information.

4 Certain types of data may not be handled by the cursor management
5 system, but instead would bypass the cursor management system and be sent
6 directly to the navigation application. These types of data may include shape
7 points. The navigation application would provide appropriate memory buffers
8 to accept the shape point information.

9 **VIII. Porting and Configuration**

10 FIG. 7 is a flow chart showing a process for compiling the source code
11 for the data access interface layer and the navigation application software
12 programs to form an executable module. In a preferred embodiment, the source
13 code for the navigation application functions is written and saved as libraries of
14 source code functions 502. These libraries are compiled to form an object code
15 module 504 for the navigation application functions. This object code is then
16 statically linked with the source code 506 for the interface library functions to
17 form an executable module 510 which is specific to a particular navigation
18 system platform. The executable module 510 may be installed on the
19 navigation system platform in various ways. For example, the executable
20 module may be copied from an additional drive slot or may be pre-installed
21 into some form of non-volatile memory in the navigation system.

22 In a preferred embodiment, the data access interface layer software
23 includes a number of tunable parameters whose values can be adjusted to
24 optimize performance on a particular platform. These parameters can be
25 adjusted as part of the porting process. The porting process includes optimizing
26 these tunable parameters for the combination of navigation application software,
27 OS, media device driver, and hardware.

28 The data access interface software library subsystems have the potential
29 to be configured either at compile time, run time, or both. Some of the
30 configuration information takes the form of a set of C code variables which
31 may reside in a separate source code module. The values of these variables can

1 affect the run time behavior of a particular subsystem. These variables can be
2 initially set to default values and adjusted in source code form. The resulting
3 optimized configuration information is then linked with the implementation of
4 the particular navigation subsystem. Additional configuration information takes
5 the form of conditional compilation statements and preprocessor defined
6 constants, which are used to optimize subsystem behavior at compile time.

7 **IX. Updating and compatibility across version changes**

8 As mentioned above, one of the advantages provided by the data access
9 interface layer is that it enables updating of the geographic database. Updating
10 can occur in several contexts. For example, in the context of new navigation
11 systems, it may be that a particular navigation system platform is offered for
12 sale over a number of years. It is desired that when the navigation system is
13 sold, the geographic database provided with the system is up-to-date. This
14 objective can be accomplished using the data access interface layer, described
15 above, by installing a storage medium upon which is stored an up-to-date
16 geographic database into the navigation system at the time of delivery to the
17 end-user customer. Furthermore, from the standpoint of the end-user, it is
18 desirable to allow the end-user to obtain updated geographic data after the
19 navigation system is installed over the lifetime of the navigation system. To
20 meet this latter objective, subscriptions may be offered that provide end-users
21 with updates on a regular basis. The data access interface layer permits the
22 end-user's system to use updated geographic data.

23 Update transactions can be handled using any of several alternatives.
24 One alternative is to update the geographic data at a central location, compile
25 an entirely new geographic database, and prepare new copies of the storage
26 medium with the new database. These new media would then be distributed to
27 end-user subscribers who then replace their old storage media with the new
28 media. This alternative can be used when the media is not rewritable, such as a
29 CD-ROM. If the medium is rewritable, such as a PCMCIA card, the new
30 geographic database can be applied directly to the old medium. Another
31 alternative is to provide updated geographic data in the form of a separate file

1 or database that contains a listing of changes relative to the version of the
2 geographic database in the end-user's system. This separate file can be located
3 on the same medium as the original geographic database (in the case of
4 re-rewritable media), or on a separate medium (in the case that the original
5 medium is read-only). Once the separate file can be accessed in the navigation
6 system, it is used together with the original geographic database. The updated
7 data from the separate file are transparently applied to the appropriate data
8 entities as they are accessed by the navigation application through the data
9 access interface layer (e.g., a "look aside"). This function 811 may be
10 implemented in the physical-to-logical subsystem 244 of the interface layer.
11 The updated geographic data can be distributed by various different means,
12 including wireless broadcast or updating stations, as disclosed in copending
13 application Serial No. 08/592,737, entitled "SYSTEM AND METHOD FOR
14 DISTRIBUTING INFORMATION FOR STORAGE MEDIA", filed January 26,
15 1996, the entire disclosure of which is incorporated by reference herein.

16 The data access interface layer not only allows the navigation system to
17 use updated geographic data, but also allows the navigation system to use
18 geographic data provided in new formats. The data access interface layer 41
19 and the logical data model can be extended to take advantage of new data
20 attributes or values that are added to the geographic database in the future as
21 requirements arise. New attributes or values can appear in the map coverage
22 data or in the physical storage format. As new map data features emerge, these
23 can be included on the geographic map data media and can be used by any
24 navigation systems that support them. Thus, the physical storage format may
25 evolve to include additional types of information beyond what it currently
26 provided. However, because the data access interface layer isolates the
27 navigation application from the physical geographic database, the navigation
28 application manufacturer need not upgrade its navigation application software
29 every time the geographic database is updated. In addition, the end-user who
30 has an installed navigation system can acquire updated geographic databases
31 over the years and be assured that they will work in his navigation system even
32 if the updated geographic databases include new types of information. Because

1 the geographic database publisher-distributor does not have to produce multiple
2 different geographic database formats for a variety of different hardware
3 platforms, the geographic databases can be made available to end-users at
4 potentially lower costs, with fewer distribution complexities, and possibly more
5 frequently, as well as with other advantages.

6 If new attributes are added to the physical storage format, or if other
7 changes to the physical storage format are implemented, the version level of the
8 physical storage format may evolve beyond the version level of the navigation
9 application program. Thus, when an updated geographic database is installed in
10 a navigation system, it is possible for the logical data model format and
11 physical storage format version levels to differ. As database version levels
12 increase, as reflected in the physical storage format, forward compatibility is
13 provided by means of the data access interface layer to maintain the viability of
14 the end user's system. Forward compatibility can be supported in this manner
15 for a long time, possibly up to ten years. Some data version changes may
16 consist of adding new attributes or values, and therefore one way that the data
17 access interface layer can accommodate new attribute or values is to filter out
18 the new data, reordering and transforming them to fit the old template.

19 Compatibility across different version levels is provided by
20 metatranslation. Metatranslation uses a set of tables for each version level.
21 Referring to FIG. 3, these metadata tables 259 reside on the storage medium 22
22 (e.g. on the CD-ROM or PCMCIA card). The metadata tables include
23 information that describes the location, size, type, and content of the various
24 data attributes and values appearing in a given database entity. In the physical-
25 to-logical data translation subsystem 244, conversion between the compressed
26 physical storage format on the medium to the decompressed intermediate format
27 usable by the query logic subsystem 210 utilizes a metadata table for each
28 physical storage format data representation. The physical storage format
29 metadata is used to extract a data element relative to the beginning of a
30 particular physical storage format entity. The data element then undergoes a
31 translation or conversion process which is controlled by the information in the
32 metadata tables.

1 Metadata tables may be located on the storage medium for each data
2 version from the earliest version to a current version. The version level of the
3 software library that forms the data access interface layer 41 is used to identify
4 the appropriate set of metadata to use.

5 The metadata tables are read from the storage medium through the
6 operating system kernel and physical devices subsystem (i.e., the operation
7 system isolation layer 302 and the media device isolation layer 300). From
8 these subsystems, the resource management subsystem 220 provides the
9 metadata to the physical-to-logical data translation subsystem 244 where the
10 metatranslation is performed. In a preferred embodiment, the metadata are
11 physically read from the medium only at initialization time. The metadata
12 tables are read into memory that is allocated from the heap on a semipermanent
13 basis, so that the metatranslation process can access the tables efficiently for
14 each data access.

15 Metadata tables can also be used to provide backward compatibility.
16 Some navigation system developers may design their systems so that the
17 navigation application software can be upgraded or updated over time after the
18 navigation system is sold. This upgradability may be offered to take advantage
19 of new features offered in the geographic database. In order to allow a newer
20 navigation application software program to use an older version of a geographic
21 database, a metadata table, for example table 813, can be provided in the new
22 version of the navigation application software. Like the metadata table
23 provided on the storage medium, the information in the metadata table provided
24 with the newer version of navigation application software is used to compare
25 the version levels of the navigation software and the physical storage format.
26 The information in these metadata tables is provided to the physical-to-logical
27 subsystem 244 at initialization time to perform any necessary attribute
28 conversions. As mentioned above, if the version levels of the navigation
29 application program and the physical storage format are the same, the metadata
30 conversion step is unnecessary and the physical-to-logical subsystem can skip
31 this conversion step.

1 **X. Further alternative embodiments**

2 The above embodiments disclose an interface layer system that can be
3 used in a navigation system. In one present embodiment, the navigation system
4 is an in-vehicle navigation system that is installed and located in an end-user's
5 vehicle. The type of vehicle may include automobiles, trucks, buses, and so on.
6 The end-users of such a system may include individuals who own or lease their
7 own vehicles, as well as fleet owners, businesses, car rental agencies, trucking
8 companies, transportation companies, and so on. In alternative embodiments,
9 the navigation system may be other-than-in-vehicle. For example, the
10 navigation system may be a hand-held unit that a person can carry on his/her
11 person. Also, the navigation system may be installed in a non-mobile location,
12 such as at a service station, car rental agency, a personal computer and so on.
13 Still further, the navigation system may be installed and used by traffic control
14 agencies, police, delivery services, and so on.

15 In a preferred embodiment, the interface layer is statically-linked to the
16 navigation application programs after the application programs are compiled in
17 order to form an executable module. In an alternative embodiment, the
18 interface layer may be dynamically linked to the navigation applications.

19 An alternative embodiment is shown in FIG. 8. In this alternative
20 embodiment, asynchronous I/O is supported by the navigation system's media
21 device driver 609. Implementation of asynchronous I/O requires that when a
22 disk I/O is submitted to the kernel, the process invoking the request can
23 continue to run and is notified when the I/O request is complete. In such an
24 alternative, a shared code version of the I/O manager submits the I/O request
25 and return control to the navigation application. When the I/O is completed, an
26 asynchronous event would be posted to a client message queue for servicing.

27 In this alternative, the I/O manager 270 may exist completely within the
28 statically-linked portion 271 of resource management subsystem 220, i.e the I/O
29 manager 270 would be linked with each of the navigation application programs
30 200. In this alternative, the I/O queue is implemented as a control structure

1 611 in the memory pool. In this case, interprocess communication techniques,
2 such as semaphore protection, are used to manage contention to the I/O queue
3 from multiple processes. The rest of the features of the interface layers systems
4 would be similar to those described above.

5 **XI. Conclusion**

6 The data access interface layer described above provides a uniform
7 interface that is incorporated in a navigation system. The data access interface
8 layer can be utilized on different navigation system platforms developed by
9 different manufacturers. The data access interface layer functions regardless of
10 the hardware platform or end-user functionality of the navigation system. The
11 data access interface layer provides a common transparent mechanism for
12 accessing geographic data stored on a physical medium. The data access
13 interface layer isolates the navigation application programs from the details of
14 the organization of the geographical data and the physical requirements of the
15 specific storage medium implementation.

16 Because the data access interface layer presents a uniform, consistent
17 view of the geographic database, navigation system developers can continue to
18 develop and enhance their navigation systems to provide new and better
19 functionality without concern about interfacing with the physical medium and
20 the storage format of the geographic database. Navigation system developers
21 can therefore focus on providing new types of navigation application programs
22 and new hardware platforms and operating systems having access to a
23 consistent geographic data interface. From the end-user's point of view, the
24 data access interface layer provides that updated geographic data can be used in
25 the end-user's navigation system, thereby reducing the risk that the navigation
26 system might become obsolete due to the geographic data version becoming out
27 of date.

28 It is intended that the foregoing detailed description be regarded as
29 illustrative rather than limiting and that it is understood that the following

1 claims including all equivalents are intended to define the scope of the
2 invention.

FIG. 1

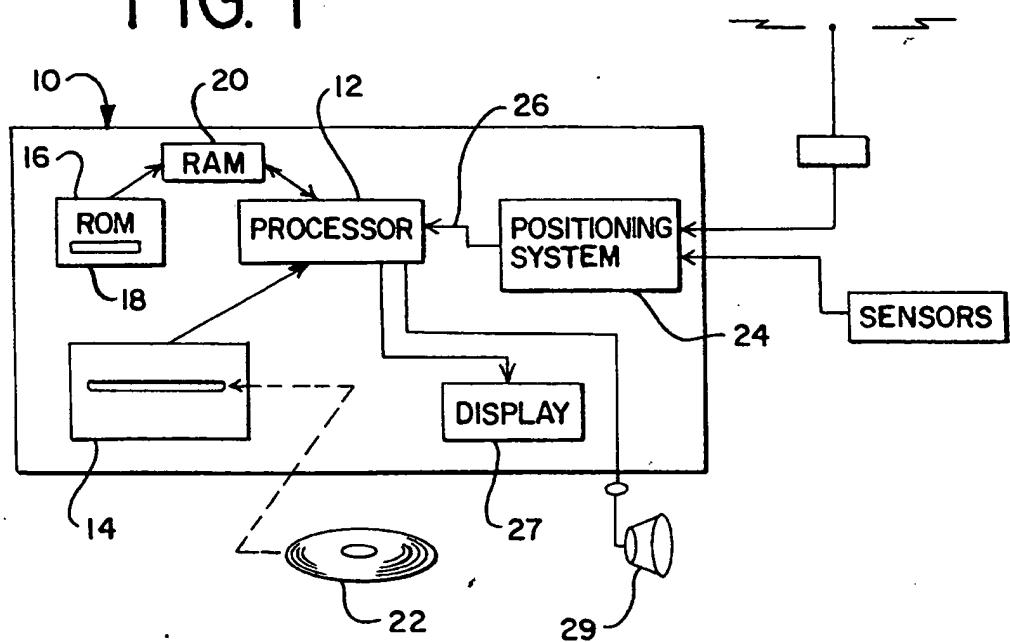


FIG. 2

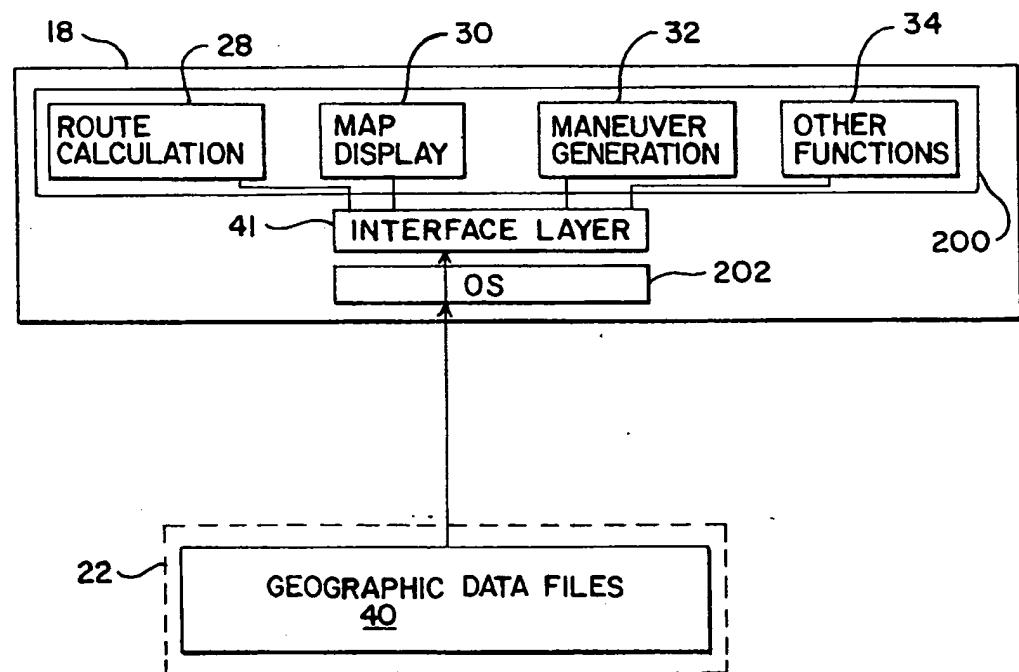
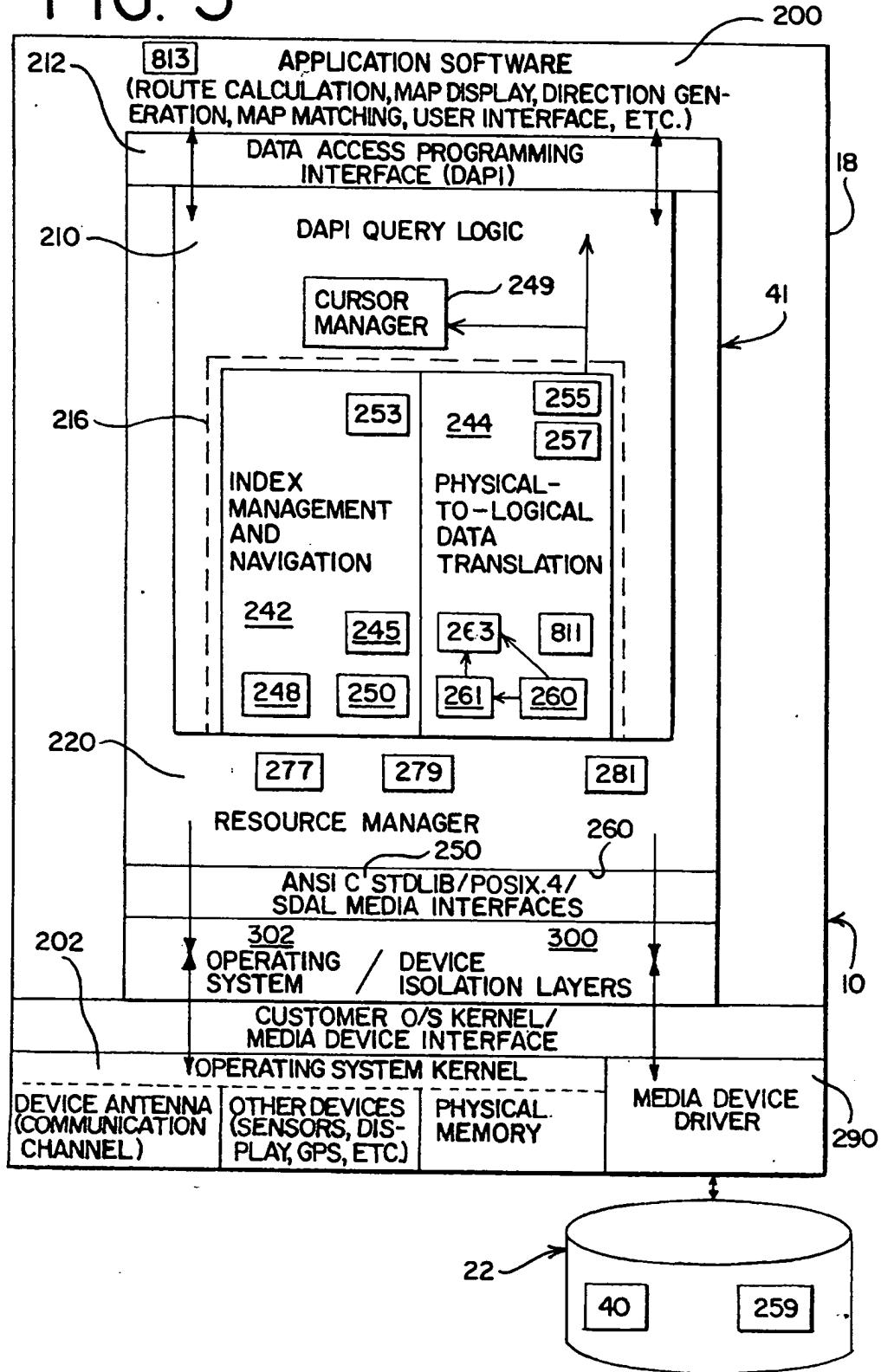


FIG. 3



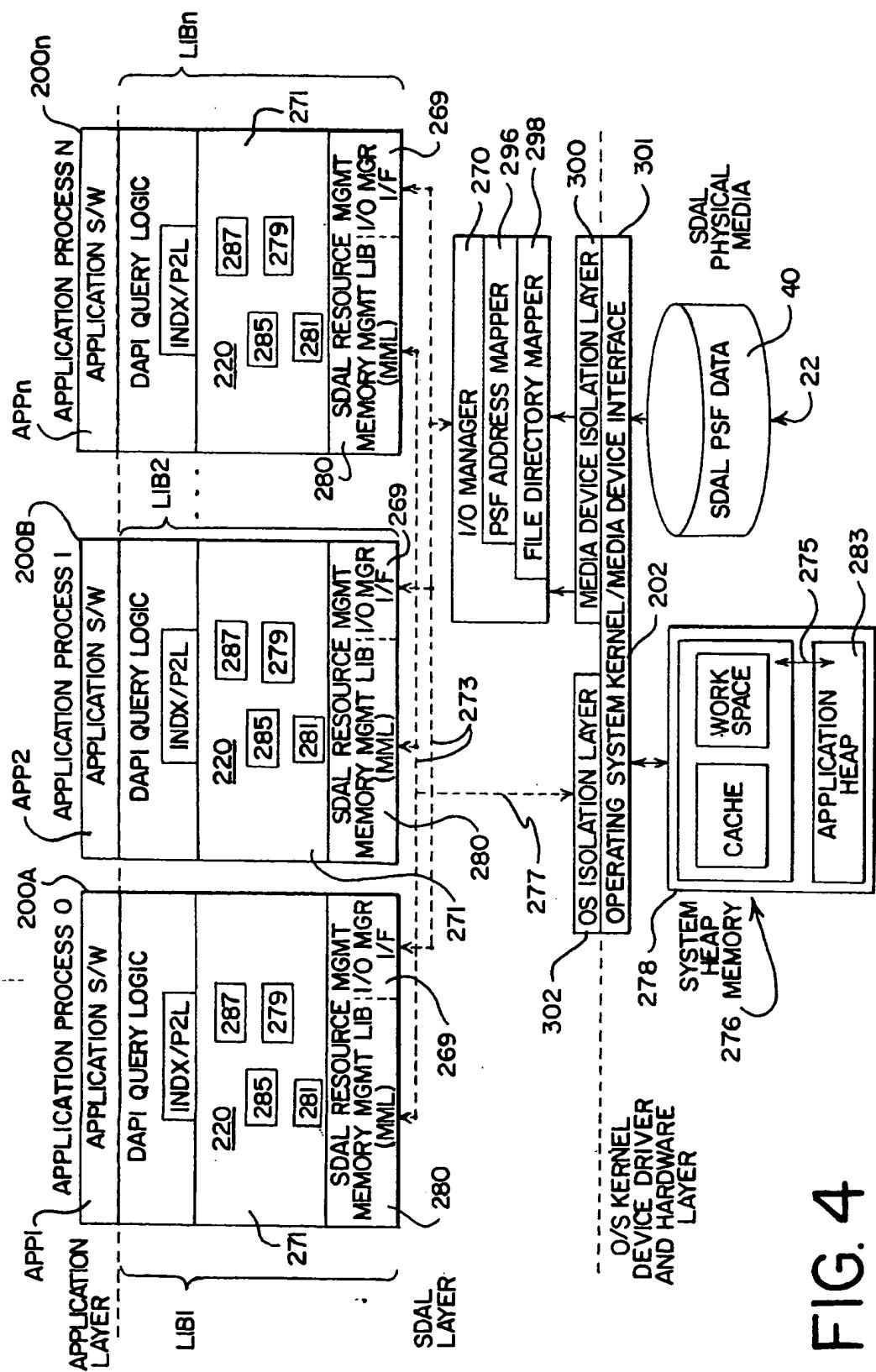


FIG. 4

FIG. 5A

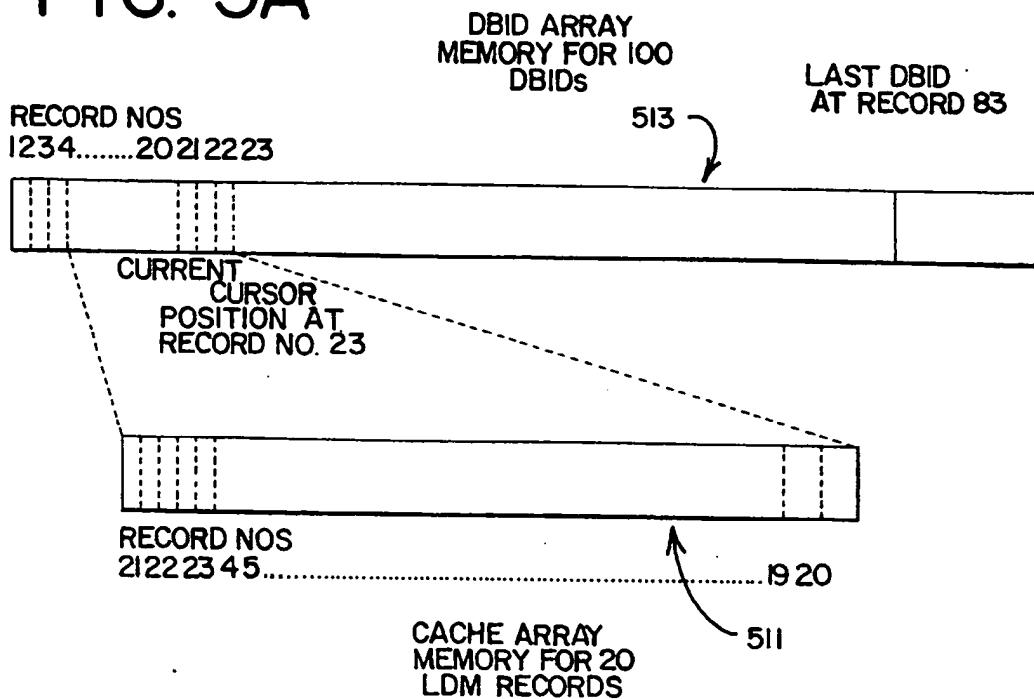


FIG. 5B

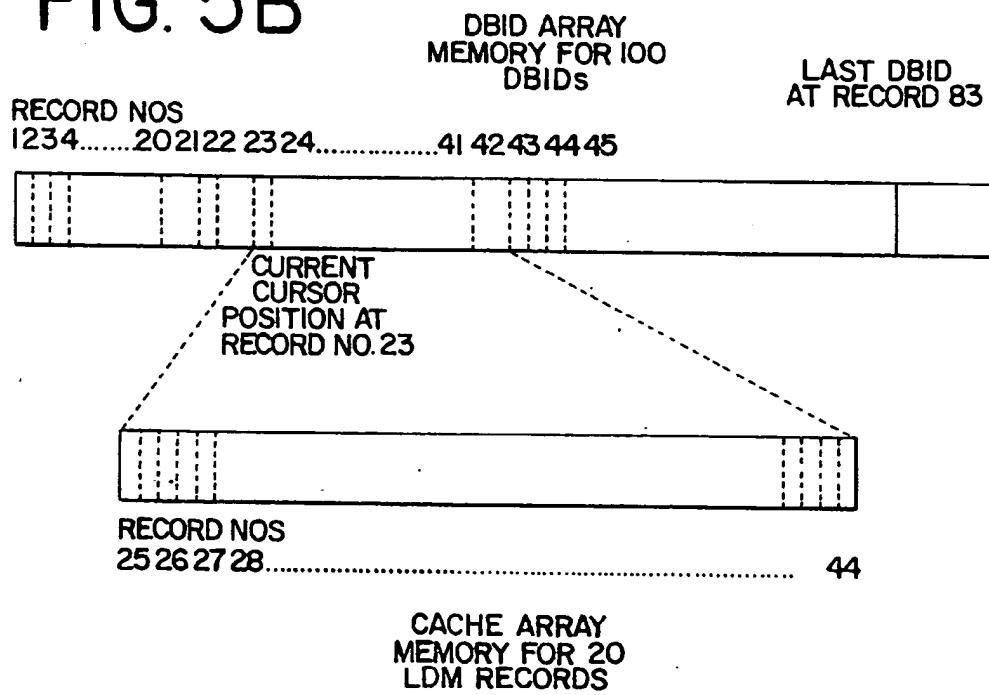


FIG. 6

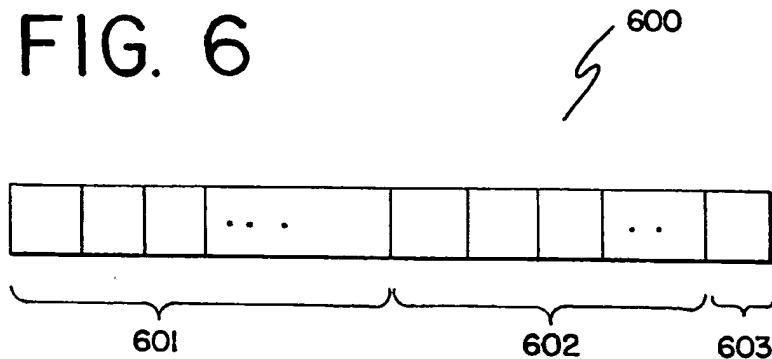
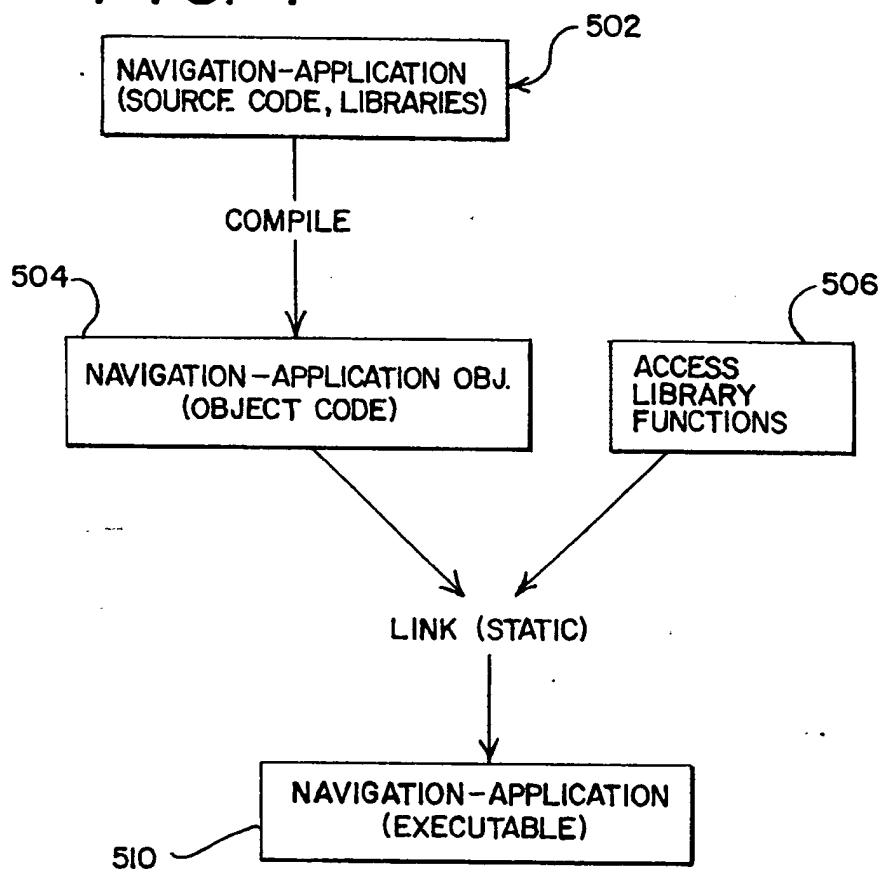
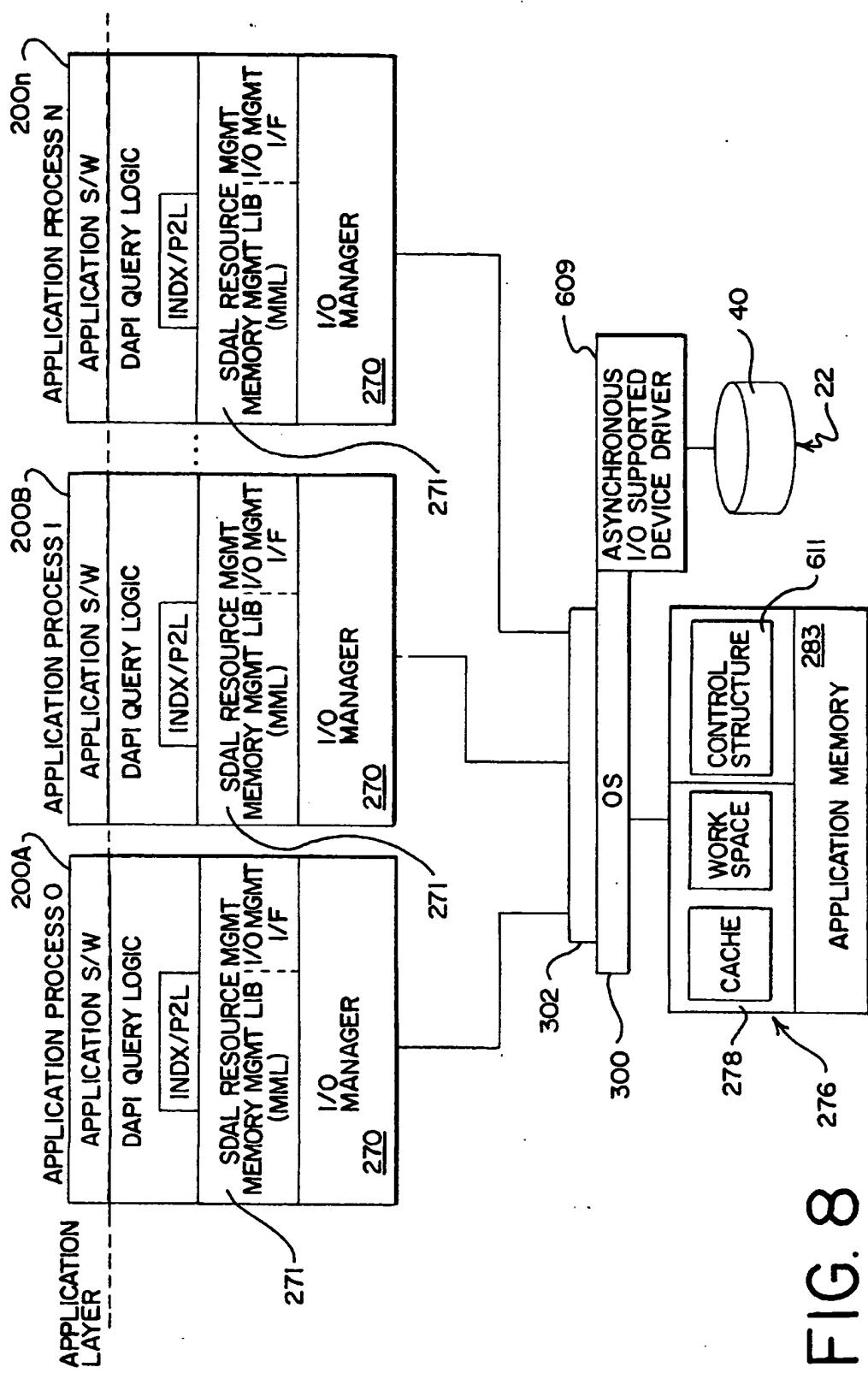


FIG. 7





8
G
E